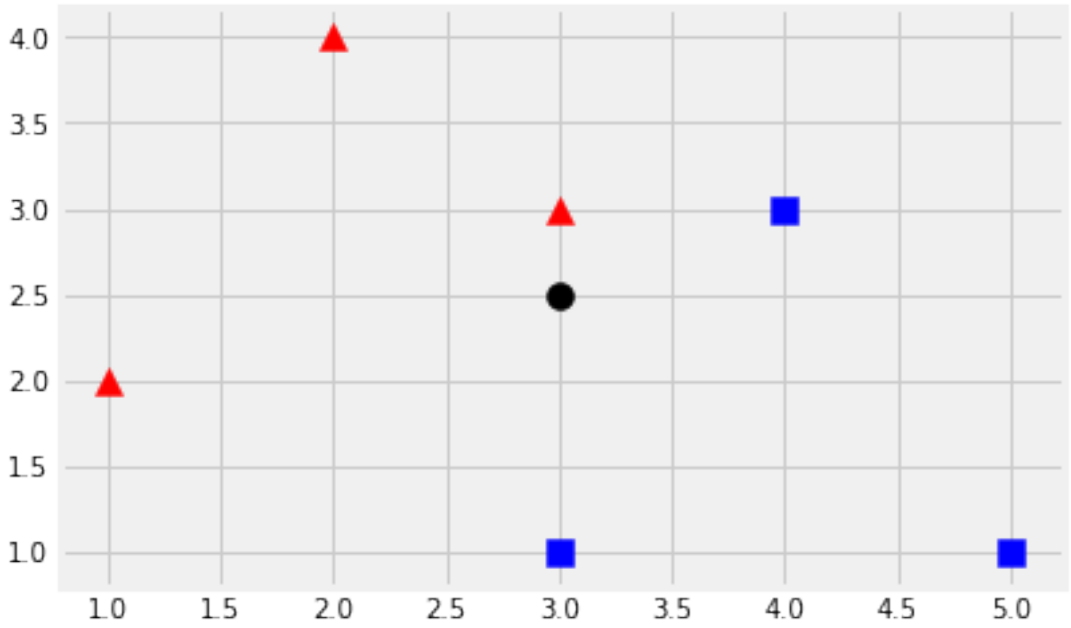
1 newpage

2 Data Science: Bridging Principles and Practice

2.1 Part 9: Machine Learning with Scikit-Learn

2.1.1 9a. The K-Nearest Neighbors Algorithm

```
In [3]: # add the new data point
        plt.scatter(x=[1, 2, 3], y=[2, 4, 3], c='r', marker='^', s=100)
        plt.scatter(x=[4, 5, 3], y=[3, 1, 1], c='b', marker='s', s=100)
        plt.scatter(x=3, y=2.5, c='k', s=100);
```



QUESTION: Using the KNN algorithm, how would this point be classified for$k = 1$,$k = 2$, and$k = 3$? Why could it be a problem to have$k$be an even number in this case?

ANSWER: for k=1, the new point would be classified as a red triangle, since the nearest point is a red triangle at (3, 3).

For k=2, it's not entirely clear how to classify the new point- the nearest neighbor is the red triangle at (3, 3), and the second nearest neighbor is a blue square at (4, 3). If both neighbors are weighted equally, then the new point would seem to be equally likely to be either class.

For k=3, the new point would be classified as a blue square. Its three nearest neighbors are the red triangle at (3, 3), the blue square at (4, 3), and the blue square at (3, 1). Because 2/3 of the neighbors are blue squares, the point is classified as a blue square.

2.1.2 9c. Using Scikit-Learn: KNN

EXERCISE: Choose explanatory and response variables.

- explanatory variables must be numerical feature column names that could help predict attrition. For this exercise, we'll be using the columns with data on the employee's monthly income, years in current role, and overtime status. The names should be strings (i.e. inside quotation marks), listed within the square brackets and separated by commas

- the response variable should be the name of the column that says whether or not an employee left by attrition, inside quotation marks

```
In [4]: # choose explanatory and response variables
        expl_vars_attrition = ["MonthlyIncome", "YearsInCurrentRole", "OverTime"]

        resp_var_attrition = "Attrition"
```

EXERCISE: Create the KNN model. Replace the ellipses with a model creation call expression, which is the model type followed by parentheses (don't put anything in the parentheses to use the default number of neighbors). We will assign the created model the name of `knn` using an equals sign.

```
In [6]: from sklearn.neighbors import KNeighborsClassifier

        # create the KNeighborsClassifier
        knn = KNeighborsClassifier()
```

EXERCISE: Fit the KNN model to the training data. Use the fit method on the model to train it. fit takes two arguments: the training feature matrix (X_train_att) and the list of classes for the training data (y_train_att).

```
In [7]: # fit the knn model to the training data
        # look at the code for the linear regression model for a hint
        knn.fit(X_train_att, y_train_att)

Out[7]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                             weights='uniform')
```

EXERCISE: Use knn.predict to make predictions for the test data (X_test_att). Check the code for the linear regression model predictions for a hint on how the code expression should look.

```
In [8]: # use the fitted knn model to predict attrition for the test data
        # look at the code for the linear regression model for a hint
        knn_predictions = knn.predict(X_test_att)
        knn_predictions

Out[8]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        ... Omitting 7 lines ...
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0])
```

EXERCISE: Use the model's score method to evaluate its accuracy on the test data. scorerequires 2 arguments: a matrix of feature data (X) and an array of response variables (y).

```
In [9]: # use the fitted knn model to score predictions for the test data
        # look at the code for the linear regression model for a hint
        knn.score(X_test_att, y_test_att)

Out[9]: 0.8061224489795918
```

QUESTION: based on the model score, how well is our classifier working? Based on the scatter plot, who are the people most likely to be mis-classified, and why?

ANSWER: Our classifier is working fairly well for a simple model of a complex problem. It has an accuracy score of about 0.806, meaning that it will correctly predict that an employee will or will not leave IBM a little over 80% of the time. If our model was guessing randomly at classifications, we'd expect to have a score of about 0.5 (50% accuracy, where the model is just as likely to guess right as it is to guess wrong).

KNN works best when the neighbors of a point have the same class as the point itself. So, based on the scatter plots, this would be any points that are mostly surrounded by points of a different shape/color. For example:

- people leaving IBM who have a high monthly income and who have worked a long time in their current role (gold circle points in the top right of the 2D plot)
- people planning to stay at IBM who have a low monthly income, have not worked long in their current role, and who are eligible for overtime (blue triangle points in the top left of the 3D plot)

```
In [1]: import gsExport
        gsExport.generateSubmission("09Machine-Learning-SOLUTIONS.nbconvert.ipynb")
```

```
Processing 09Machine-Learning-SOLUTIONS.nbconvert.ipynb
Generated notebook and autograded
Attempting to compile LaTeX
Finished generating PDF


<IPython.core.display.HTML object>
```