A graph with a person and arrow going down

Description automatically generated

**Churn Prediction Model**

**Telecom churn** refers to the phenomenon where customers leave or cancel their service with a telecom provider, either switching to a competitor or discontinuing the service entirely. This is a major concern for telecom companies as acquiring new customers can be more expensive than retaining existing ones.

A **predictive machine learning model** can be useful by analyzing customer data (such as usage patterns, service complaints, payment history, and more) to identify patterns and predict which customers are likely to churn. By flagging at-risk customers, the company can proactively offer targeted retention strategies, like personalized offers or improved service, to prevent them from leaving. This can help reduce churn rates and increase customer loyalty.

In this exercise you will analyze the *churn.all* dataset and build a prediction model.

1. Load the provided dataset and create a dataframe called *churn\_df*, where the following variables are selected:

*['account\_length', 'total\_day\_charge', 'total\_eve\_charge', 'total\_night\_charge', 'total\_intl\_charge', '*number\_customer\_service\_calls*', 'churned']*

1. Convert the variable churn to a numeric format, where False. corresponds to 0 and True. to 1.
2. Perform all necessary data pre-processing steps discussed in previous classes in order to prepare the data to build the prediction model.
3. **k-Nearest Neighbors: Fit**

In this exercise, you will build your first classification model using the *churn\_df* dataset.

The target, "churn", needs to be a single column with the same number of observations as the feature data.

"account\_length" and "customer\_service\_calls" are treated as features because account length indicates customer loyalty, and frequent customer service calls may signal dissatisfaction, both of which can be good predictors of churn.

Instructions:

* Import KNeighborsClassifier from sklearn.neighbors.
* Instantiate a KNeighborsClassifier called knn with 6 neighbors.
* Fit the classifier to the data using the *.fit()* method.

Fill in the code:

# Import KNeighborsClassifier

from \_\_\_\_.\_\_\_\_ import \_\_\_\_

y = churn\_df["churn"].values

X = churn\_df[["account\_length", "customer\_service\_calls"]].values

# Create a KNN classifier with 6 neighbors

knn = \_\_\_\_(\_\_\_\_=\_\_\_\_)

# Fit the classifier to the data

knn.\_\_\_\_(\_\_\_\_, \_\_\_\_)

1. **k-Nearest Neighbors: Predict**

Now you have fit a KNN classifier, you can use it to predict the label of new data points. All available data was used for training, however, fortunately, there are new observations available. These are stored in a data frame called *X\_new* and defined as follows:

X\_new = np.array([[30.0, 17.5],

[107.0, 24.1],

[213.0, 10.9]])

The model knn, which you created and fit the data in the last exercise, has been preloaded for you. You will use your classifier to predict the labels of a set of new data points:

* Create *y\_pred* by predicting the target values of the unseen features *X\_new* using the knn model.
* Print the predicted labels for the set of predictions.

# Predict the labels for the X\_new

y\_pred = \_\_\_\_

# Print the predictions

print("Predictions: {}".format(\_\_\_\_))

1. **Train/test split + computing accuracy**

It's time to practice splitting your data into training and test sets with the *churn\_df* dataset!

NumPy arrays have been created for you containing the features as X and the target variable as y.

Instructions:

* Import *train\_test\_split* from sklearn.model\_selection.
* Split X and y into training and test sets, setting test\_size equal to 20%, random\_state to 42, and ensuring the target label proportions reflect that of the original dataset.
* Fit the knn model to the training data.
* Compute and print the model's accuracy for the test data.

# Import the module

from \_\_\_\_ import \_\_\_\_

X = churn\_df.drop("churn", axis=1).values

y = churn\_df["churn"].values

# Split into training and test sets

X\_train, X\_test, y\_train, y\_test = \_\_\_\_(\_\_\_\_, \_\_\_\_, test\_size=\_\_\_\_, random\_state=\_\_\_\_, stratify=\_\_\_\_)

knn = KNeighborsClassifier(n\_neighbors=5)

# Fit the classifier to the training data

\_\_\_\_

# Print the accuracy

print(knn.score(\_\_\_\_, \_\_\_\_))

1. **Overfitting and underfitting**

Interpreting model complexity is a great way to evaluate supervised learning performance. Your aim is to produce a model that can interpret the relationship between features and the target variable, as well as generalize well when exposed to new observations.

The training and test sets were created from the *churn\_df* dataset and set as X\_train, X\_test, y\_train, and y\_test.

In addition, KNeighborsClassifier has been imported for you along with numpy as np.

Instructions:

* Create neighbors as a numpy array of values from 1 up to and including 12.
* Instantiate a KNeighborsClassifier, with the number of neighbors equal to the neighbor iterator.
* Fit the model to the training data.
* Calculate accuracy scores for the training set and test set separately using the .score() method, and assign the results to the train\_accuracies and test\_accuracies dictionaries, respectively, utilizing the neighbor iterator as the index.

# Create neighbors

neighbors = np.arange(\_\_\_\_, \_\_\_\_)

train\_accuracies = {}

test\_accuracies = {}

for neighbor in neighbors:

# Set up a KNN Classifier

knn = \_\_\_\_(\_\_\_\_=\_\_\_\_)

# Fit the model

knn.\_\_\_\_(\_\_\_\_, \_\_\_\_)

# Compute accuracy

train\_accuracies[\_\_\_\_] = knn.\_\_\_\_(\_\_\_\_, \_\_\_\_)

test\_accuracies[\_\_\_\_] = knn.\_\_\_\_(\_\_\_\_, \_\_\_\_)

print(neighbors, '\n', train\_accuracies, '\n', test\_accuracies)

1. **Visualizing model complexity**

Now you have calculated the accuracy of the KNN model on the training and test sets using various values of *n\_neighbors*, you can create a model complexity curve to visualize how performance changes as the model becomes less complex!

The variables neighbors, train\_accuracies, and test\_accuracies, which you generated in the previous exercise, have all been preloaded for you. You will plot the results to aid in finding the optimal number of neighbors for your model.

Instructions:

* Add a title "KNN: Varying Number of Neighbors".
* Plot the .values() method of train\_accuracies on the y-axis against neighbors on the x-axis, with a label of "Training Accuracy".
* Plot the .values() method of test\_accuracies on the y-axis against neighbors on the x-axis, with a label of "Testing Accuracy".
* Display the plot.

# Add a title

plt.title("\_\_\_\_")

# Plot training accuracies

plt.plot(\_\_\_\_, \_\_\_\_, label="\_\_\_\_")

# Plot test accuracies

plt.plot(\_\_\_\_, \_\_\_\_, label="\_\_\_\_")

plt.legend()

plt.xlabel("Number of Neighbors")

plt.ylabel("Accuracy")

# Display the plot

\_\_\_\_

1. **Repeat with another ML algorithm**

Repeat the previous procedures using the Naïve Bayes Algorithm (GaussianNB from scikit-learn).