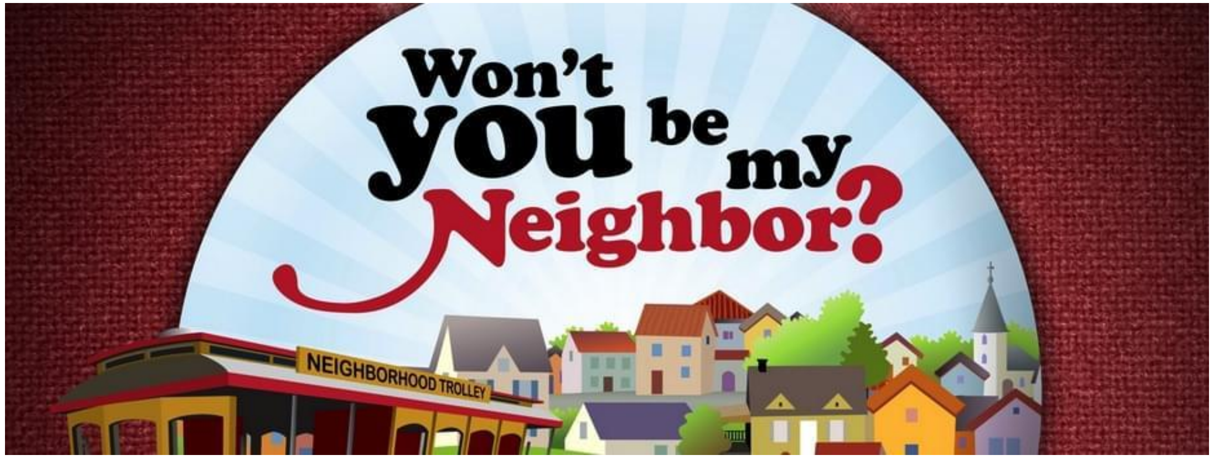
YData: Introduction to Data Science

Lecture 22: Classification

Overview



Quick review of KNN classifier

Cross-validation

Other classifiers

Building our own KNN classifier

If there is time: feature normalization

Project timeline

Sunday, November 17^{th}

- Projects are due on Gradescope at 11pm
- Email a pdf of your project to your peer reviewers
    - A list of whose paper you will review is posted on Canvas
    - Fill out the draft reflection on Canvas

Sunday, November 24^{th}

- Jupyter notebook files with your reviews need to be sent to the authors
- A template for doing your review has been posted

Sunday, December 8^{th}

- Project is due on Gradescope
    - Add peer reviews to the Appendix of your project

Homework 9 has been posted

- It is due December 1^{st}

Extra office hours tomorrow (11/15) will be from 1-2pm

Review of Classification

Prediction: regression and clasification

We “learn” a function f

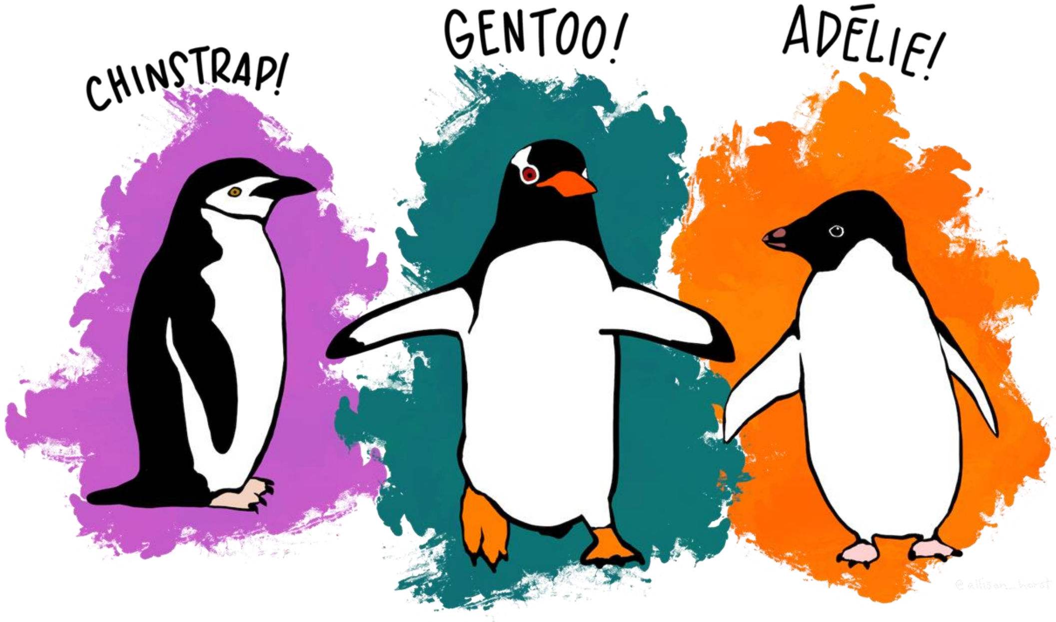
- f(**x**) $\longrightarrow$ y

Input: **x** is a data vector of "features"

Output:

- Regression: output is a real number ($y \in R$)
- Classification: output is a categorical variable y_k

Example: Penguin species



What are the features and labels in this task?

- Labels (y): Chinstrap, Gentoo, Adelie
- Features (X): Flipper length, bill length, body mass, …

Classifiers

Classifiers take a list/array of features X, and return a predicted label y

There are many different types of classifiers

- Decision Trees, Support Vector Machines, Logistic regression, Neural Networks, etc.

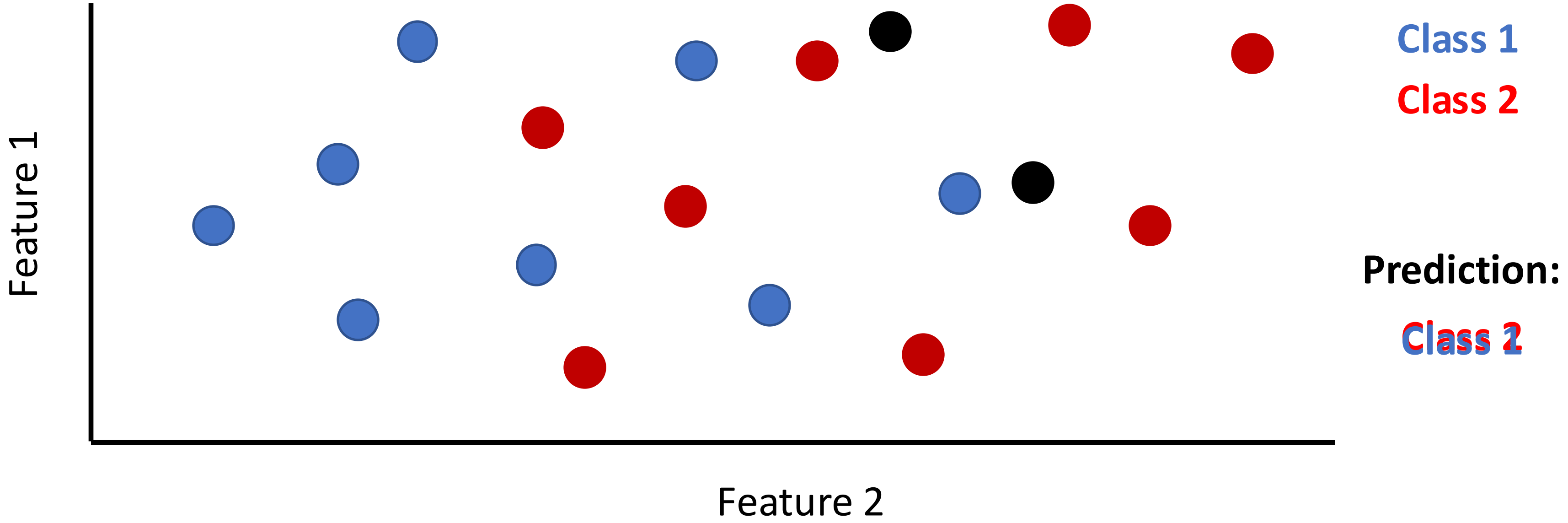
The classification process always involves two steps:

1. **Training** the classifier to learn the relationship between features (X) and labels (y) (from many examples)
    - This is done using the .fit() method in scikit-learn
2. **Using** the classifier to predict a label y, from features X for a new data point
    - This is done using the .predict() method in scikit-learn

K-Nearest Neighbor Classifier (KNN)

Training the classifier: Store all the features with their labels

Making predictions: The label of closest k training points is returned



KNN classifiers using scikit-learn

We can fit and evaluate the performance of a KNN classifier using:

```
knn = KNeighborsClassifier(n_neighbors = 1)        # construct a classifier

knn.fit(X_features, y_labels)       # train the classifier

penguin_preditions = knn.predict(X_penguin_features)  # make predictions

np.mean(penguin_preditions == y_penguin_labels)       # get accuracy
```

Evaluation

Review: overfitting

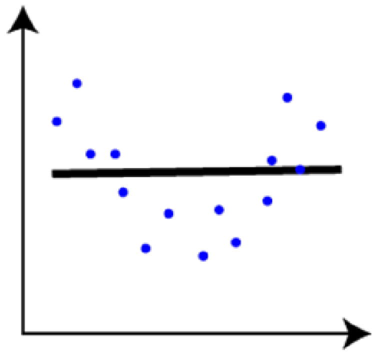
Overfitting occurs when our classifier matches too close to the training data and doesn't capture the true underlying patterns

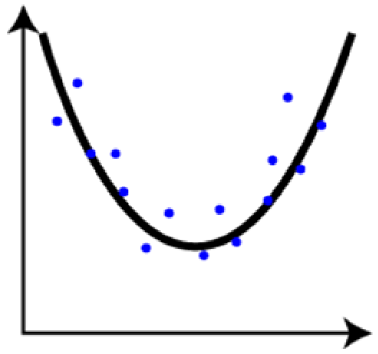
If our classifier has overfit to the training data then:

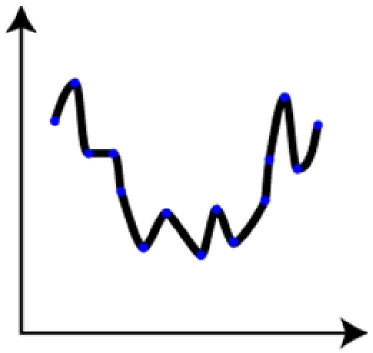
a. We might not have a realistic estimate of how accurate its predictions will be on new data

b. There might be a better classifier that would not over-fit to the data and thus can make better predictions

What we really want to estimate is how well the classifier will make predictions on new data, which is called the **generalization (or test) accuracy**







Overfitting song…

Review: Cross-validation

To assess how accurate the predictions of a classifier are, we need to split our data into a training set and test set

We fit the classifier on the training set

- i.e., we learn the relationship between labels (y) and features (x) on the training set

We assess the classifier's performance on the test set

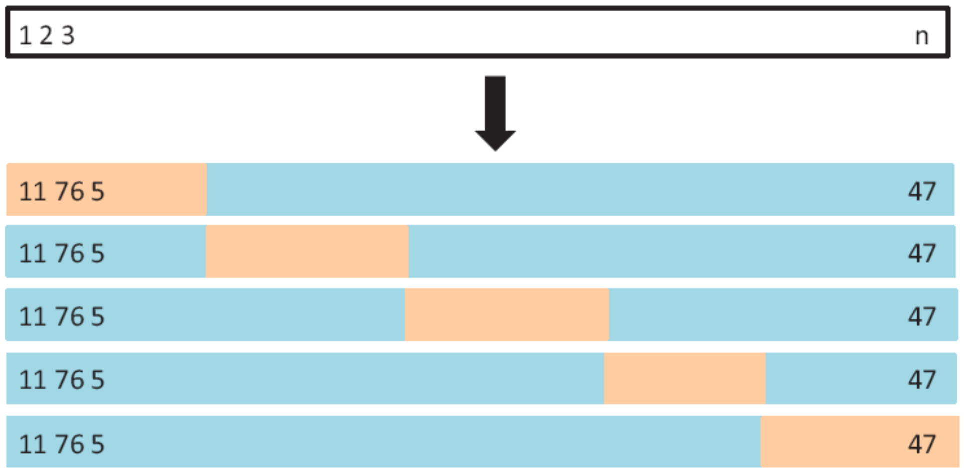
The process of fitting your classifier on a training set and evaluation your classifier on a test set is called cross-validation

Cross-validation can help prevent overfitting!

k-fold cross-validation

k-fold cross-validation

- Split the data into k parts
- Train on k-1 of these parts and test on the left out part
- Repeat this process for all k parts
- Average the prediction accuracies to get a final estimate of the generalization error

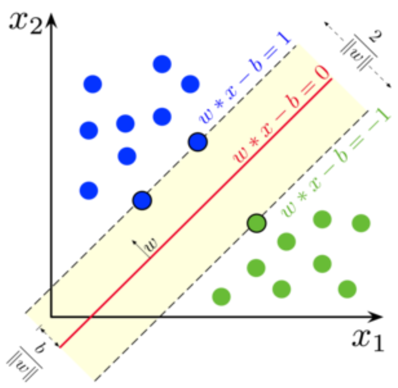


Let's try this in Jupyter!

Other classifiers

There are many other classification algorithms such as:

- Support Vector Machines (SVM)
- Decision Trees/Random Forests
- Deep Neural Networks
- etc.

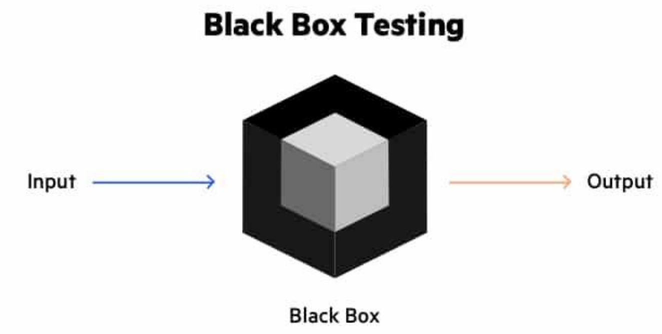


Scikit-learn makes it easy to try out different classifiers get their cross-validation performance

```
svm = LinearSVC()

scores = cross_val_score(svm, X_features,  y_labels, cv = 5)

scores.mean()
```



Let's quickly try this in Jupyter!

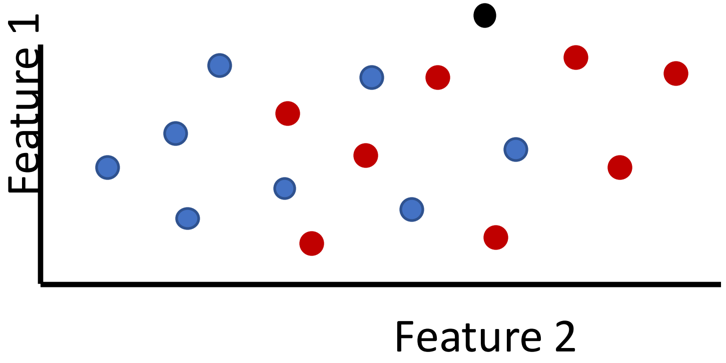
Building a KNN classifier

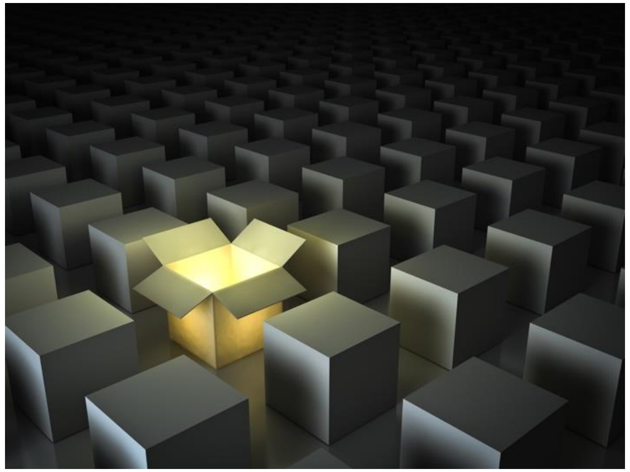
Building the KNN classifier

So far we have used a KNN classifier

- and we have some idea of how it works

Let's now see if we can write to to implement the classifier ourselves…





Steps to build a KNN classifier

We build our KNN classifier by creating a series of functions…

1. euclid_dist(x1, x2)
   - Calculates the Euclidean distance between two points x1 and x2

$$D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}$$

2. get_labels_and_distances(test_point, X_train_features, y_train_labels)
   - Finds the distance between a test point and all the training points

3. classify_point(test_point, k, X_train_features, y_train_labels)
   - Classifies one test point by returning the majority label of the k closest points

4. classify_all_test_data(X_test_data, k, X_train_features, y_train_labels)
   - Classifiers all test points

Let's continue exploring this in Jupyter!

Feature normalization

Review: Distance between two points

Two features x and y: $D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$

Three features x, y, and z: $D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}$

- And so on for more features…

It's important the features are standardized

- If not, features that typically have larger values will dominate the distance measurement

Feature normalization

In order to deal with features that are measured on very different scales, we can normalize the features

With a z-score normalization, we normalize each feature to have:

- A mean of 0
- A standard deviation of 1

We can do this in Python using:

```
scalar = StandardScaler()
scalar.fit(X_train)
X_train_transformed = scalar.transform(X_train)
X_test_transformed = scalar.transform(X_test)
```

bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
46.1	15.1	215.0	5100.0
37.3	17.8	191.0	3350.0
51.3	18.2	197.0	3750.0
39.5	16.7	178.0	3250.0
48.7	15.1	222.0	5350.0

$\bar{x}$

s

$$z_i = \frac{x_i - \bar{x}}{s}$$

Feature normalization

To avoid overfitting ("data leakage") we can:

- Calculate the mean and standard deviation on the training
- Apply these means and standard deviations to normalize the training and test sets

To do this in a cross-validation loop, we can use a pipeline:

```
scalar = StandardScaler()
knn = KNeighborsClassifier(n_neighbors = 1)
cv = KFold(n_splits=5)

pipeline = Pipeline([('transformer', scalar), ('estimator', knn)])

scores = cross_val_score(pipeline, X_penguin_features, y_penguin_labels, cv = cv)
```

Let's explore this in Jupyter!