

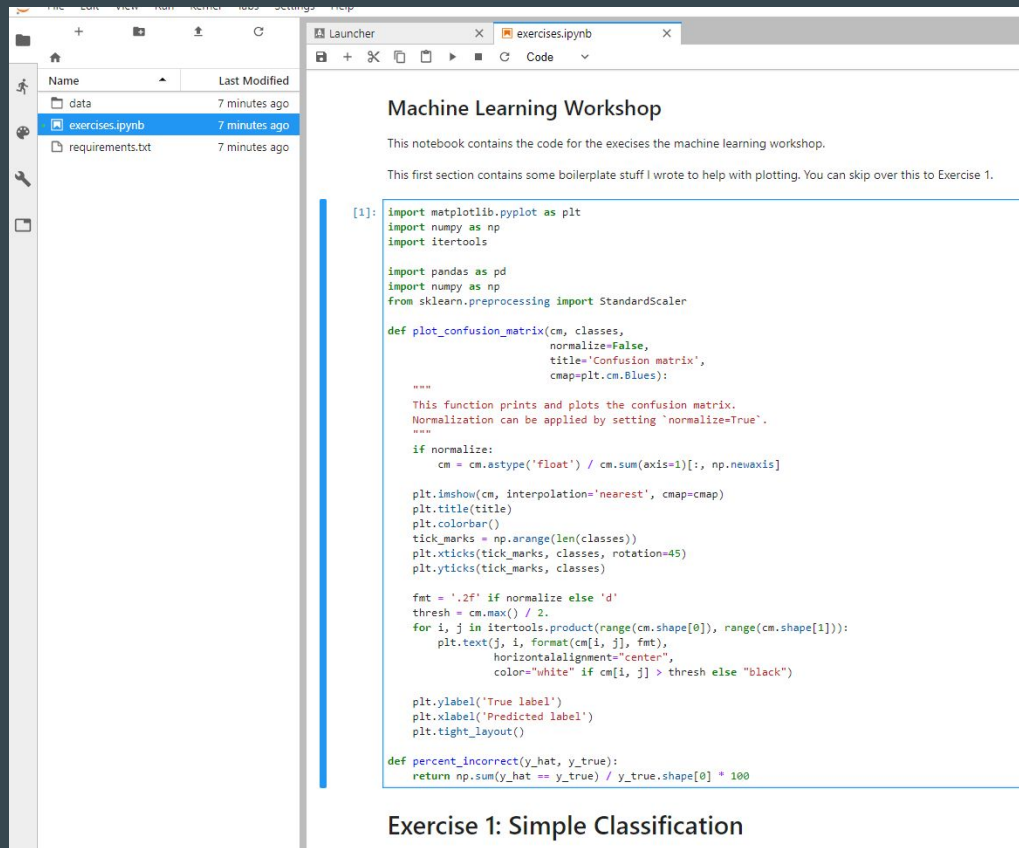
Machine Learning

Go to:

<https://mybinder.org/v2/gh/samueljackson92/ml-workshop/master?urlpath=lab>

Binder

- Interactive Jupyter Lab environment
- Open exercises.ipynb
- Can run code in each of the boxes
- Data should already be there



The screenshot displays the Binder Jupyter Lab interface. On the left, a file explorer shows a directory with files: 'data', 'exercises.ipynb' (highlighted), and 'requirements.txt'. The main area is a code editor for 'exercises.ipynb'. It contains a Python notebook with a title 'Machine Learning Workshop' and two text blocks. The first text block states: 'This notebook contains the code for the exercises the machine learning workshop.' The second text block states: 'This first section contains some boilerplate stuff I wrote to help with plotting. You can skip over this to Exercise 1.' Below the text is a code cell with the following Python code:

```
[1]: import matplotlib.pyplot as plt
import numpy as np
import itertools

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    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]

    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()

def percent_incorrect(y_hat, y_true):
    return np.sum(y_hat != y_true) / y_true.shape[0] * 100
```

Below the code cell, the text 'Exercise 1: Simple Classification' is visible.

What We'll Cover:

- What is machine learning?
- Using scikit-learn for:
 - Classification
 - Clustering
 - Model Selection
- 30 seconds of Pandas

What We Won't Cover:

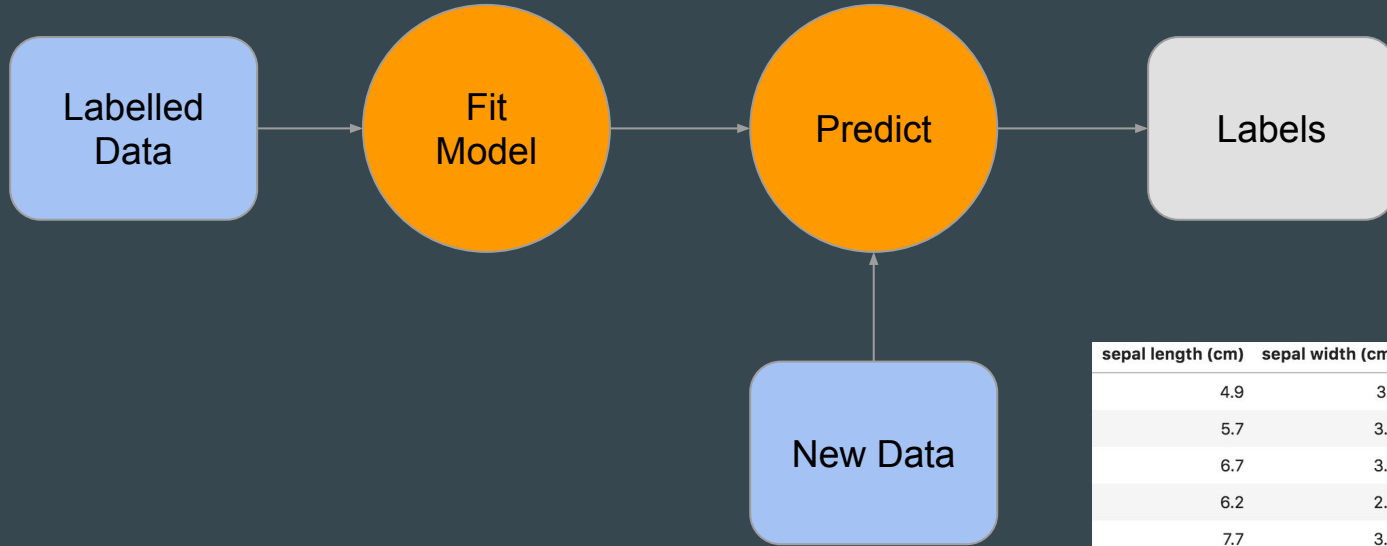
- Lots of math
- Neural Networks
- Tensorflow
- ...



Machine Learning definition

“A set of methods that can automatically detect patterns in data, then use the uncovered pattern to predict future data”

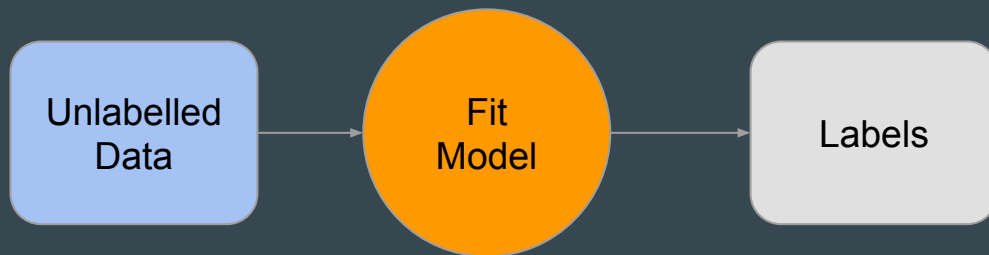
Types of Machine Learning: Supervised



sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	labels
4.9	3.1	1.5	0.1	0
5.7	3.0	4.2	1.2	1
6.7	3.3	5.7	2.5	2
6.2	2.8	4.8	1.8	2
7.7	3.8	6.7	2.2	2
5.5	2.6	4.4	1.2	1
5.6	3.0	4.5	1.5	1
6.1	2.8	4.7	1.2	1
5.5	3.5	1.3	0.2	0
6.7	3.3	5.7	2.1	2

- Classification: which one is this?
- Regression: predicting a value.

Types of Machine Learning: Unsupervised



- Clustering: group together “similar” data into one label.

Model Evaluation - Supervised

- How good are our predictions?
- How do we know?
- Training/test split:
 - Split data into training and test set
 - Train only on the test set
 - Compare predictions to known label
 - There are other strategies...
- KFold Cross Validation:
 - Split data into K-folds
 - Use one fold as “test” and train on others
 - Repeat

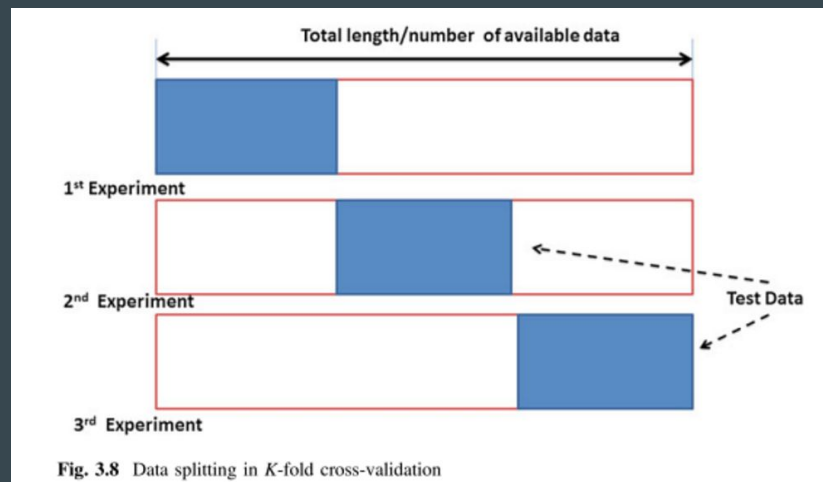
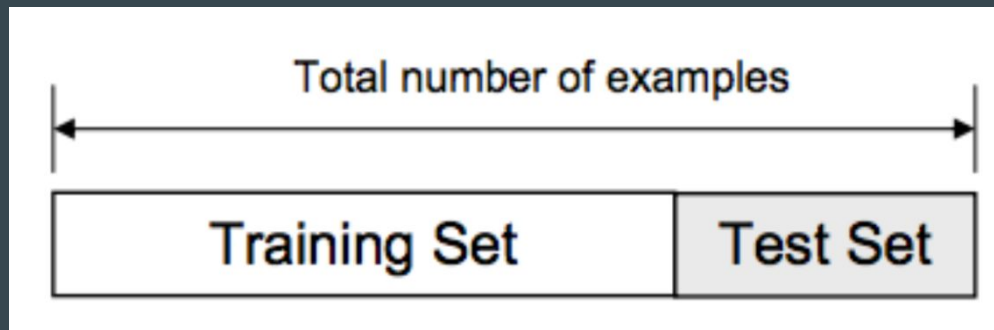
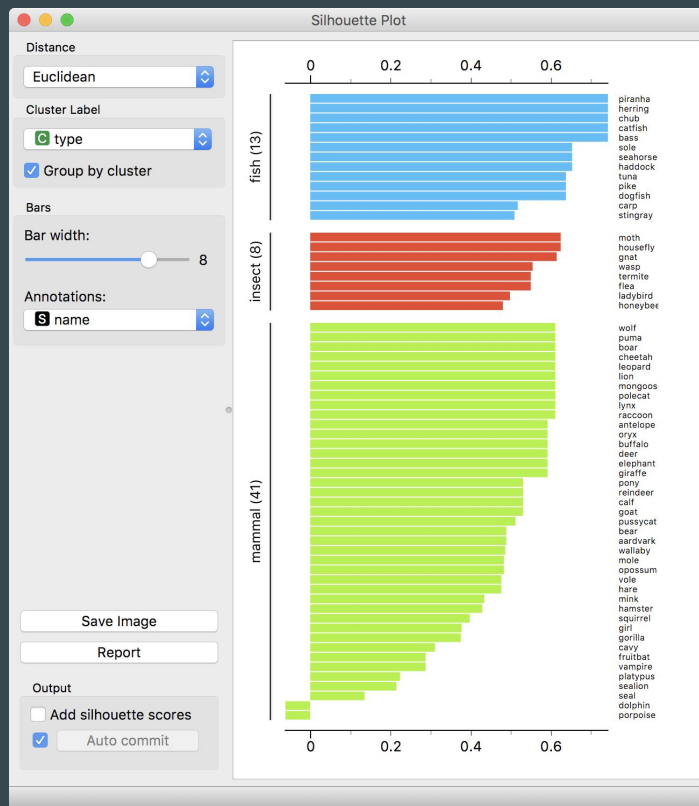


Fig. 3.8 Data splitting in K-fold cross-validation

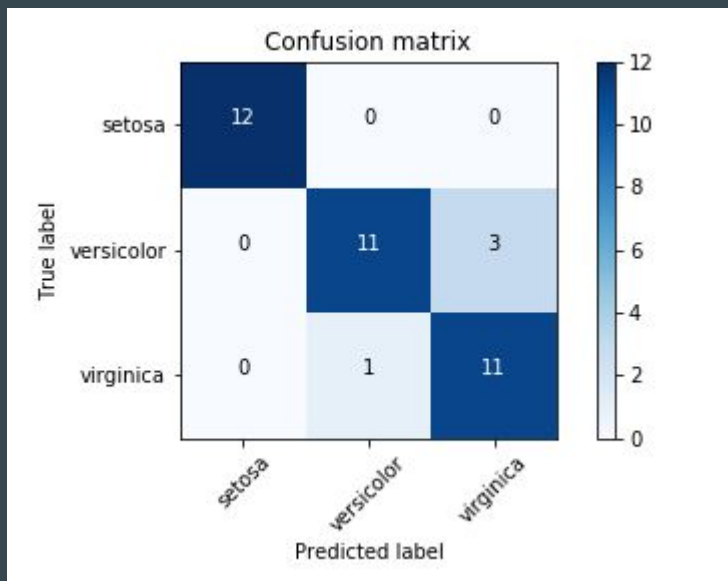
Model Evaluation - Unsupervised

- Harder to know
- We don't have any labels!
- Silhouette score
 - a measure of how close each point in one cluster is to points in the neighboring clusters
 - Higher == More consistent



Model Evaluation

- So how do we compare labels?
- Simple percentage of what we got wrong (accuracy)
- Confusion matrix



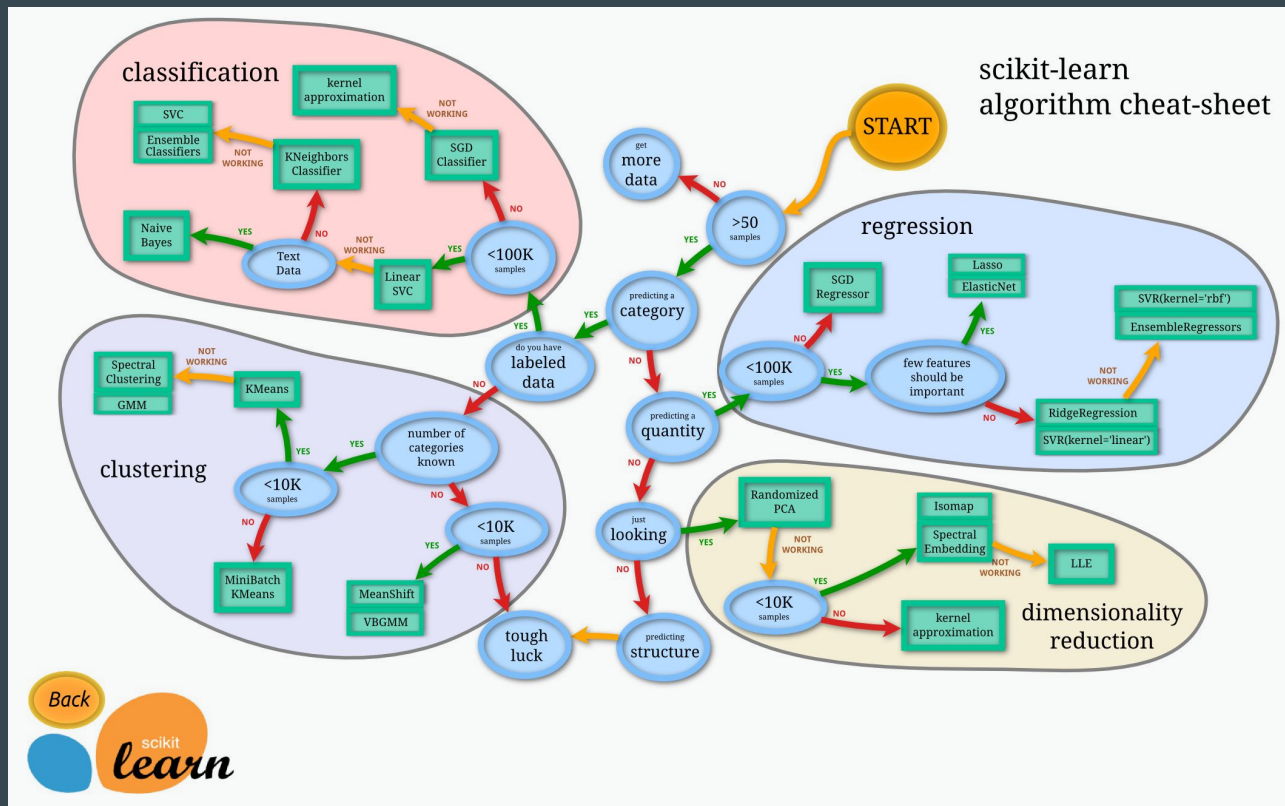
Scikit-learn

- Python library for machine learning
- Consists of a bunch of *Estimator* classes.
 - These have methods *fit*, and *predict*

```
from sklearn import linear_model
from sklearn import datasets

iris = datasets.load_iris()
clf = linear_model.LogisticRegression(solver='lbfgs')
X, y = iris.data, iris.target
clf.fit(X, y)
y_predicted = clf.predict(X)
```

Scikit-learn Estimators



Model Selection

```
from sklearn.model_selection import train_test_split

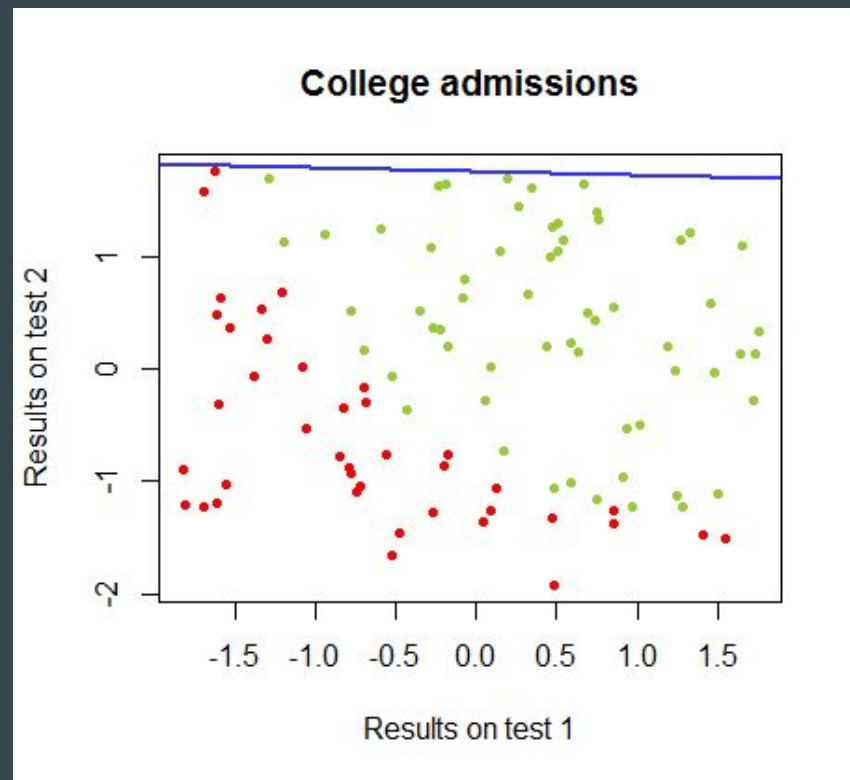
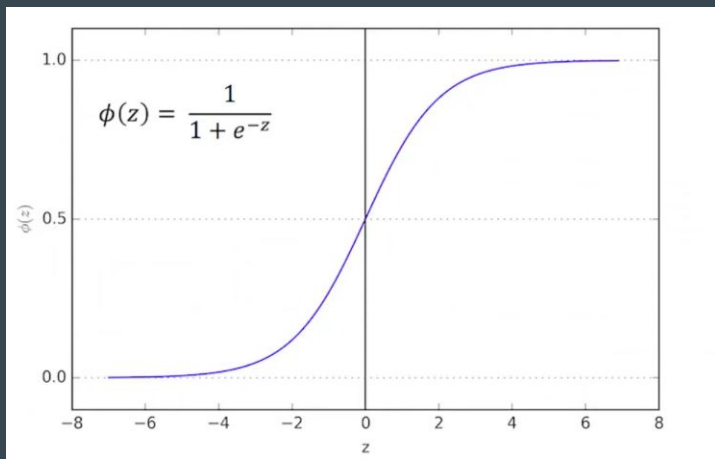
X_train, X_test, y_train, y_test = \
    train_test_split(iris.data, iris.target, test_size=0.25)

...

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_predicted, y_test)
```

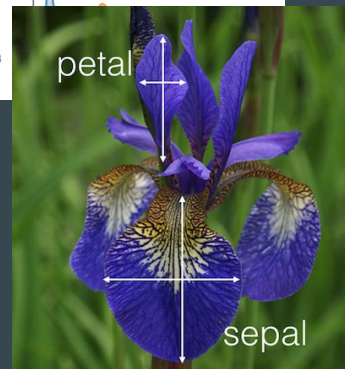
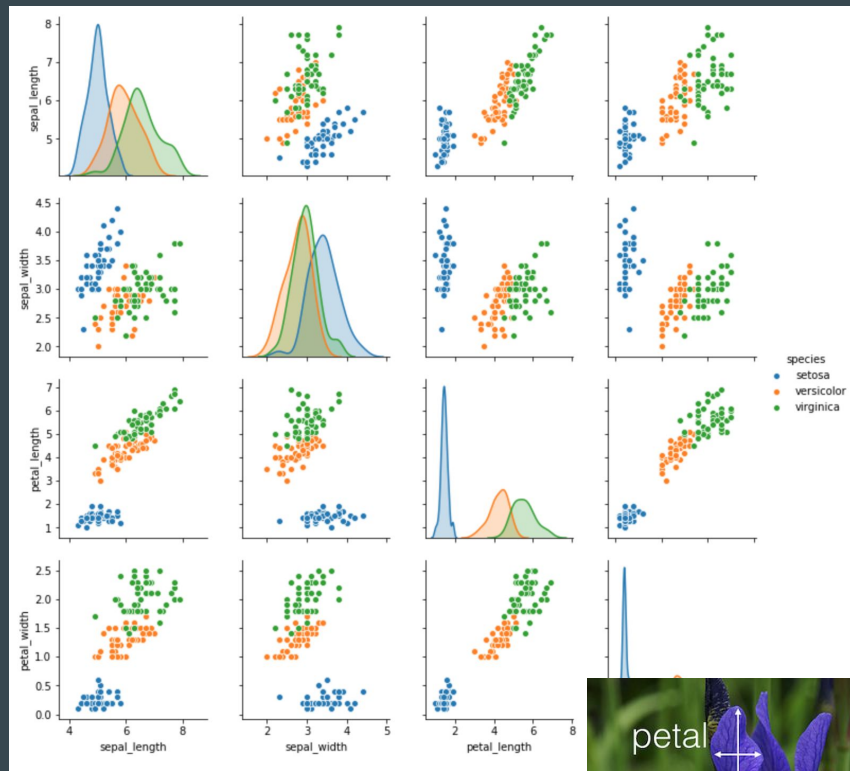
Logistic Regression

- Try to find a line that *separates* the data
- Then for each point
 - Use distance from line compute probability of class



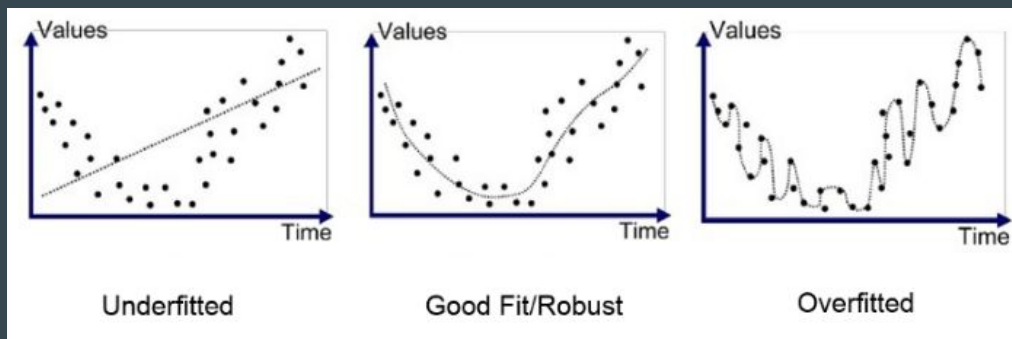
Exercise 1: Classifying Flowers

- The Iris Dataset has a 3 classes
 - Setosa, versicolor, virginica
- Run exercise 1 to train the classifier
 - Check what percentage correct do you get?
 - Run it multiple times, does it change?
 - Why? What could we do instead?
- Change the train/test split value
 - Change it to .97
 - What happens to the results?
- Change the code to use KFold cross validation
- Change the Estimator to use *SVC* from the *svm* module.
 - Repeat the above.



Underfitting & Overfitting

- Underfitting - Model cannot capture the underlying trend of the data
 - Try a more complex model, probably with more parameters
- Overfitting - Model fits too well and starts fitting to the noise in the data
 - Try a simpler model
 - Try getting more data
 - Try regularizing your model ...

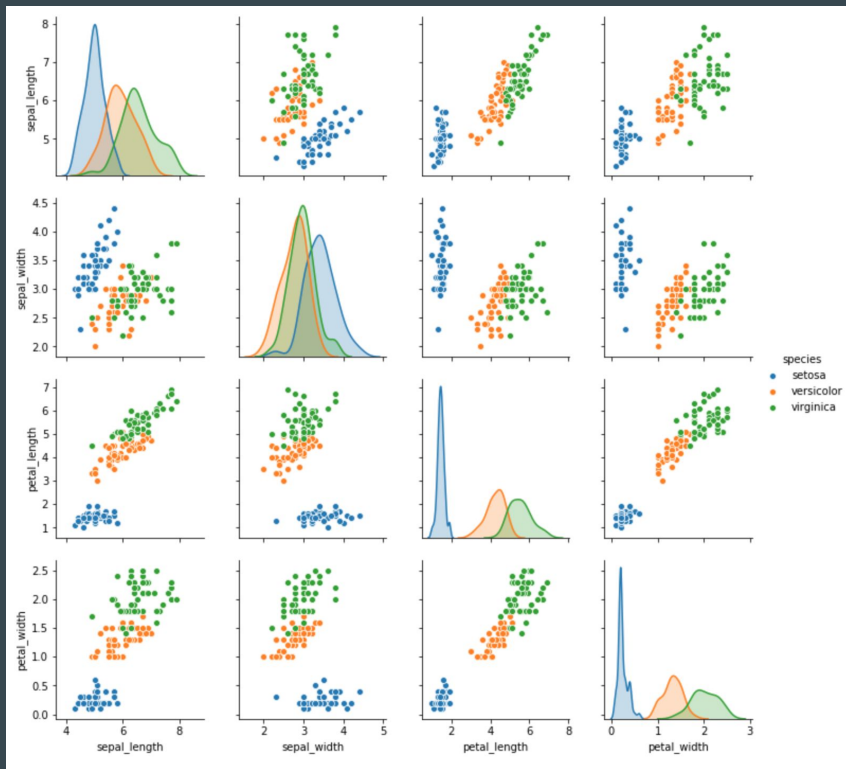


Regularization

- Add a penalty term to the cost function of the model
- L2 regularisation
 - Aka: Ridge regression.
 - Penalises very large weights.
 - Sklearn has *sklearn.linear_model.Ridge*
- **Exercise 2: Underfitting & Overfitting**
 - Polynomial model for a cosine wave.
 - Play with the number of degrees (this increases model complexity)
 - Play with the number of number of samples
 - Try swapping the model to ridge regression

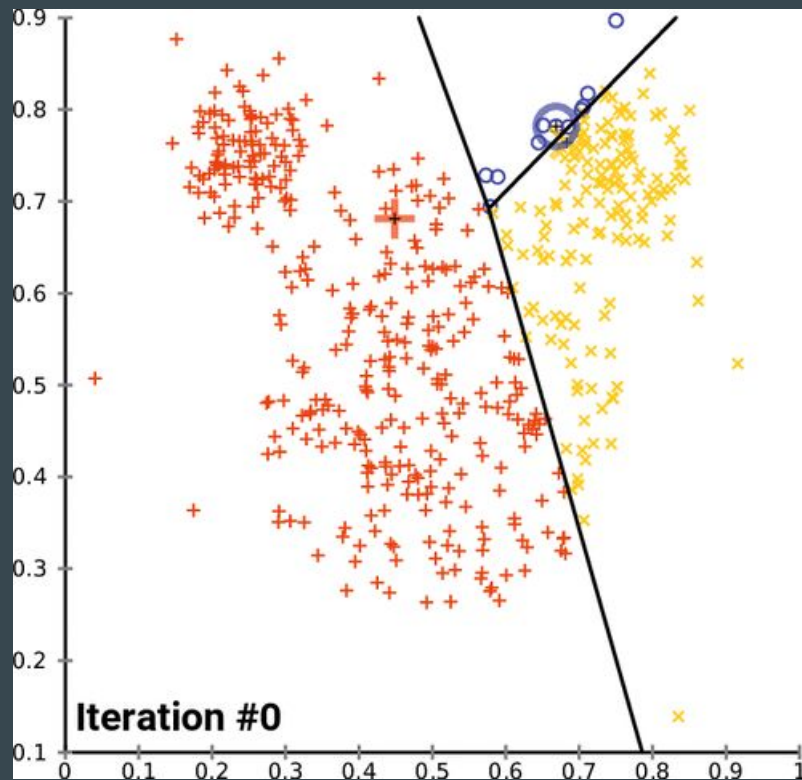
Clustering

- What if we didn't have the labels?
- The classes form clusters in the data
 - *Virginica* and *versicolor* overlap a little
- Cluster can infer structure without labels



K-Means Classifier

- Pick k points randomly
- Calculate mean distance from center points to every other point
- Classify points by whatever center is closest
- Calculate mean of those clusters
- Use mean as new center
- Rinse and repeat



Exercise 3: Clustering

- Write some code to cluster the Iris dataset
 - Using the KMeans Classifier
- Hint: cluster classifiers have a method called *fit_predict* to fit and predict in one step
- Try a different clustering technique:
 - <https://scikit-learn.org/stable/modules/clustering.html>
- Try measuring the consistency of the cluster with the silhouette score

Aside: 30 seconds of Pandas

- Awesome library for handling raw data.
- Allows us to create *DataFrames* which store data in table like structures.

```
import pandas as pd
df = pd.read_csv("data/titanic/train.csv")
df.head() # get the top n rows
df.tail() # get the bottom n rows
df['Age'] # get the column with the heading "Age"
df.drop("Name", axis=1) # drop the column called name
df.values # get the entries as a numpy matrix
df.fillna('Age': 0) # fill all NaN entries in the Age column with 0.
```

Exercise 4: Titanic Data

- For the final exercise we'll try a [Kaggle Competition](#) dataset
- Given a set of (labelled) data try and predict who survives on the titanic
- Play with different classifiers
- Play with different parameters
- Investigate what the variables mean?
 - Can we make new ones?
 - Representation is key!



You may also like:

- SciML Seminars
 - 12:30 - 13:30 Weekly.
 - 1hr presentation on a variety of ML topics (with free pizza!)
 - Workshops on practical ML are coming soon.
 - <https://indico.stfc.ac.uk/category/15/>
- Slides & code for this session:
 - <https://github.com/samueljackson92/ml-workshop>

Thank You