Intermediate Data Science

Introduction to Machine Learning

Joanna Bieri DATA201

Important Information

- Email: joanna_bieri@redlands.edu
- Office Hours take place in Duke 209 – Office Hours Schedule
- Class Website
- Syllabus

What is Machine Learning

https://en.wikipedia.org/wiki/Machine_learning

Machine learning (ML) is a field of study in artificial intelligence concerned with the development and study of statistical algorithms that can learn from data and generalise to unseen data, and thus perform tasks without explicit instructions.

Broadly speaking it is a field that was built out of statistical modeling. Even though there has been a boom in machine learning in the past few years, it is a field that has actually been around since the early 1900.

Machine Learning and Data Science

In the data science lifecycle you often move toward modeling and prediction after your initial EDA (exploratory data analysis). You have looked at the data and noticed patterns and start to wonder "can I predict something using this data?". In some cases you will use machine learning as part of this process.

Some examples include:

- Linear or logistic regression
- Time series and forecasting
- Pattern recognition
- Dimensionality reduction
- Neural networks and Classifiers

ML Process

1. Acquire and clean the data
2. Exploratory data analysis
3. Test - Train - Validate data split
4. Data wrangling - feature engineering
5. Model Training
6. Hyperparameter Tuning
7. Model Testing
8. Model Deployment or Publication

Check your installs:

```
Python version: 3.13.5 | packaged by conda-forge | (main, Jun
pandas version: 2.3.2
matplotlib version: 3.10.5
NumPy version: 2.3.2
SciPy version: 1.16.2
IPython version: 9.5.0
scikit-learn version: 1.7.2
```

Check your installs:

If you are missing any of these packages then please install them!

```
!conda install -y pacakge_name
```

Example ML project

We are going to explore a start to finish mini-machine learning project. This is like your "Hello World" introduction. As we move through the rest of the semester we will go into much more detail about model selection, training, etc.

Classification

One thing that machine learning models tend to be very good at is classification tasks. Classification is when a model takes input variables for an observation and figures out how to classify the observation from that data. For example, given a photo is it a cat or a dog?

The Data

The Iris dataset contains measurements of iris flowers from three different species. The dataset was first introduced by Sir Ronald A. Fisher in 1936. The measurements themselves were originally collected by Edgar Anderson, a botanist who studied the morphology of iris flowers from three related species.

The Data

Number of samples: 150 Number of features or variables: 4 (all numeric and continuous)

Features:

- sepal length (in cm)
- sepal width (in cm)
- petal length (in cm)
- petal width (in cm)

Target (species):

- Iris setosa
- Iris versicolor
- Iris virginica

The Data

We want to see if we can use the Features (variables) to predict the Target or the species of iris.

Load the data

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

Explore the data

This data is already in pretty good shape. You will see from the .describe() that there are no nans in any of the 150 observations.

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length
       'petal width (cm)', 'species'],
      dtype='object')
```

Explore the data

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal widt
count	150.000000	150.000000	150.000000	150.00000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Explore the data

From looking at the value_counts() we see our data is well **balanced**. Balanced data is important in prediction. If we only had, say, three or four observations of the setosa species we likely would not be very accurate in predicting that class.

Explore the data

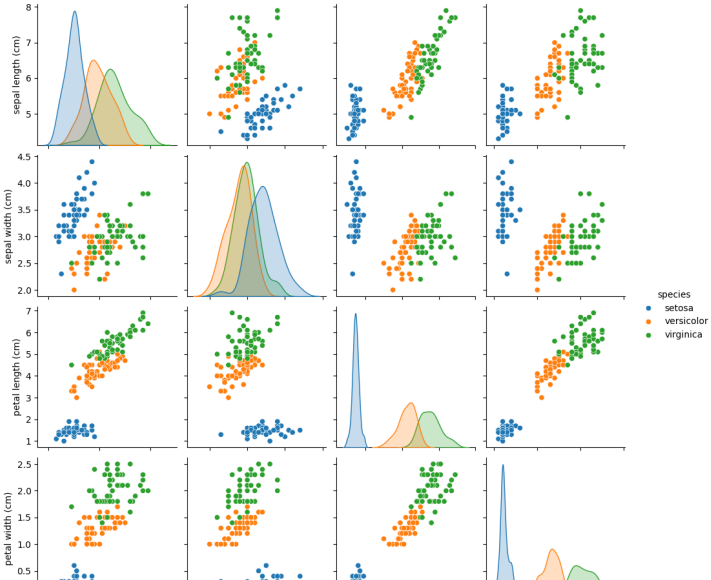
```
species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

Explore the data

We also want to look at some visualizations. Does it seem reasonable that the variables we have could be used for classification? Are there strange outliers?

Here we will do a pairplot and color it by the thing we want to predict ('species')

Explore the data



Train - Test Split

Before we get too deep into the analysis, but after we are fairly certain that our data is clean and could be usefull for classification, we want to do a train test split.

WHY SPLIT?

If you trained a prediction model using all the data and then test it using the same data there is a real concern that the model will just memorize the data. It will get really great at predicting the data you have, but will be useless at predicting new, future, data. This is just like taking an exam!

WHY SPLIT?

- If you had a teacher who gave you the whole exam (all the data) and then said, learn this. The best way to do that is to memorize the exam. Any you would get really good at taking that exam. But how good would you be at applying that knowledge to new scenarios? Probably not great!

WHY SPLIT?

- Now instead imagine you have a teacher that gives you lots of practice problems (a training set) and then choose a new set of problems on the exam for you to apply your knowledge to (a test set). If you can pass the test, then it is fairly likely that you will be able to apply your knowledge to new scenarios and memorizing the training set would not help.

WHY SPLIT?

So we split our data into Testing, Training, and sometimes Validation sets:

Training - about 80% of the data that the model gets to see when learning. (homework) **Validation** - about 10% of the data that the scientists uses to tune the model (like a practice exam) **Testing** - about 10% of the data that is the real text to see how accurate your model actually was (a final exam)

You want to be the best teacher you can be!

Train - Test Split

The `train_test_split` function from `sklearn.model_selection` helps us separate our data randomly. Here is how it's used:

```
X_train,
X_test,
y_train,
y_test = train_test_split(df[data_cols],
                          df[target_cols],
                          test_size=0.20,
                          random_state=random_seed)
```

Train - Test Split

You send in as many data sets as you want. Here we send in our features or (variables) the X data, and our target (predictions or species) the y data. In genera we want a function that tells us

$$y = F(X)$$

where we send in X and get out y. We can do optional arguments, here test_size $= .2$ says to use 20% of the data for testing, and setting a random_seed makes the results reproducible.

Train - Test Split

```
from sklearn.model_selection import train_test_split

data_cols = ['sepal length (cm)',
             'sepal width (cm)',
             'petal length (cm)',
             'petal width (cm)']
target_cols = ['species']

random_seed = 42
X_train, X_test, y_train, y_test =
                train_test_split(df[data_cols],
                                 df[target_cols],
                                 test_size=0.20,
                                 random_state=random_seed)
```

Always Check Your Shapes!

Data shape mismatch errors are very common and very frustrating.
You may as well check to see that the data shapes are what you expect.

Always Check Your Shapes!

First we see that the X data has 4 columns and this matches our four input variables. We see that training has 120 observations and testing has 30. This means 20% of the data really did end up in testing and 80% in training. If we look at the y data it has one column - just the species we want to predict.

```
X_train shape: (120, 4)
y_train shape: (120, 1)
X_test shape: (30, 4)
y_test shape: (30, 1)
```

Always Check Your Shapes!

Often you have to reshape because the ML algorithms expect the data in a certain format. In our case we want our training data to be interpreted as a list of predictions, no extra columns. So we reshape it:

```
y_test = y_test.to_numpy().reshape(-1)
y_train = y_train.to_numpy().reshape(-1)
```

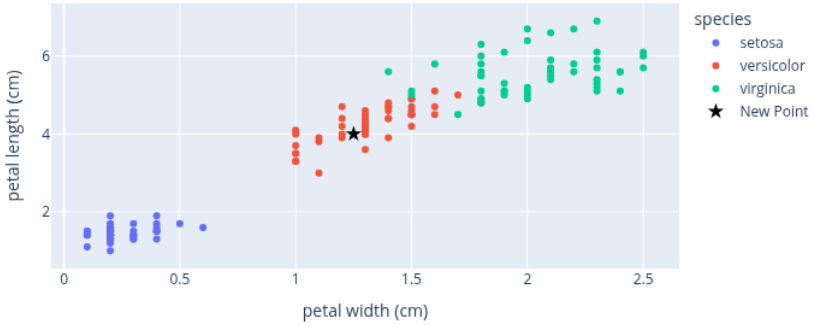
```
y_train shape: (120,)
y_test shape: (30,)
```

Model Selection

At this point we could choose a wide range of possible models. We will do a very simple model called K-Nearest Neighbors (KNN). KNN is one of the simplest ways for a computer to make predictions based on data. When the computer sees a new example, it looks at the K most similar examples it has already seen - its "nearest neighbors". If most of those neighbors belong to a certain group, the computer guesses that the new example belongs to that group too. For instance, if most nearby flowers are iris setosa, it will predict the new flower is also setosa.

Model Selection

Let's see this in action in 2-dimensions.



Model Selection

If we look at the nearest neighbors around the new point we see that it is very likely a member of Versicolor. We can choose how many neighbors to consider. I often start with just 1 or 2. The distance to the nearest neighbor can be calculated quite a few ways, as the crow flies, taxi cab on a grid, other.

Define the model in Python

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Train the Model

For KNN this is a simple as storing the training data and species types so that when it gets a new observation it has something to compare to.

```
knn.fit(X_train, y_train)
```

n_neighbors	1
weights	'uniform'
algorithm	'auto'
leaf_size	30
p	2
metric	'minkowski'
metric_params	None
n_jobs	None

KNeighborsClassifier Important Parameters Explained

`n_neighbors` (default = 5) - The K in KNN - the number of nearest neighbors the algorithm looks at when making a prediction.

- Example: `n_neighbors=3` - it looks at the 3 closest points.

KNeighborsClassifier Important Parameters Explained

`weights` (default = 'uniform') - Determines how neighbors are counted:

- `'uniform'` - all neighbors are equally important.
- `'distance'` - closer neighbors count more than farther ones.

KNeighborsClassifier Important Parameters Explained

`p` (default = 2) - The power parameter for the Minkowski distance:
- `p = 1` - Manhattan distance
- `p = 2` - Euclidean distance

KNeighborsClassifier Important Parameters Explained

`metric` (default = 'minkowski') - The distance metric used to measure "closeness":

- `'euclidean'` - straight-line distance
- `'manhattan'` - city-block distance
- `'minkowski'` - general formula (p = 1 $\rightarrow$ Manhattan, p = 2 $\rightarrow$ Euclidean)

Try a Prediction

Now that the model is trained you can see what happens in a prediction. Here we will make up a data point (a pretend flower) and see what the models does with this information.

```
X_new = pd.DataFrame([[5, 2.9, 1, 0.2]],columns=df.keys()[0:4]
prediction = knn.predict(X_new)
```

```
X_new.shape: (1, 4)
```

```
Prediction: ['setosa']
```

Great the model is giving at least reasonable outputs. It if had an error our give an answer of 72, or "dog" or something we would know we already had a problem.

Model Testing (or Validation)

Now we want to see how our model does on the exams. We are now giving it data that it did not see during the training phase to check if it can do a good job predicting that data. When we call

```
knn.predict()
```

we are sending X data into our model $F(x)$ to see what it returns as the y value, in this case a category.

Model Testing (or Validation)

```
y_pred = knn.predict(X_test)
```

```
Test set predictions:
 ['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicol
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virgi
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'set
```

How does this compare?

We can look at the real answer

```
y_test
```

```
array(['versicolor', 'setosa', 'virginica', 'versicolor', 'ver
       'setosa', 'versicolor', 'virginica', 'versicolor', 'ver
       'virginica', 'setosa', 'setosa', 'setosa', 'setosa', 'v
       'virginica', 'versicolor', 'versicolor', 'virginica', '
       'virginica', 'setosa', 'virginica', 'virginica', 'virgi
       'virginica', 'virginica', 'setosa', 'setosa'], dtype=ob
```

Accuracy and Scoring

We usually want to actually score our model. How accurate was it on average?

```
# Using numpy
np.mean(y_pred == y_test)

# Using the built in .score() method
knn.score(X_test, y_test)
```

```
Numy Test set score: 1.00

KNN Test set score: 1.00
```