Exploring Classifiers with Python Scikit-learn — Iris Dataset

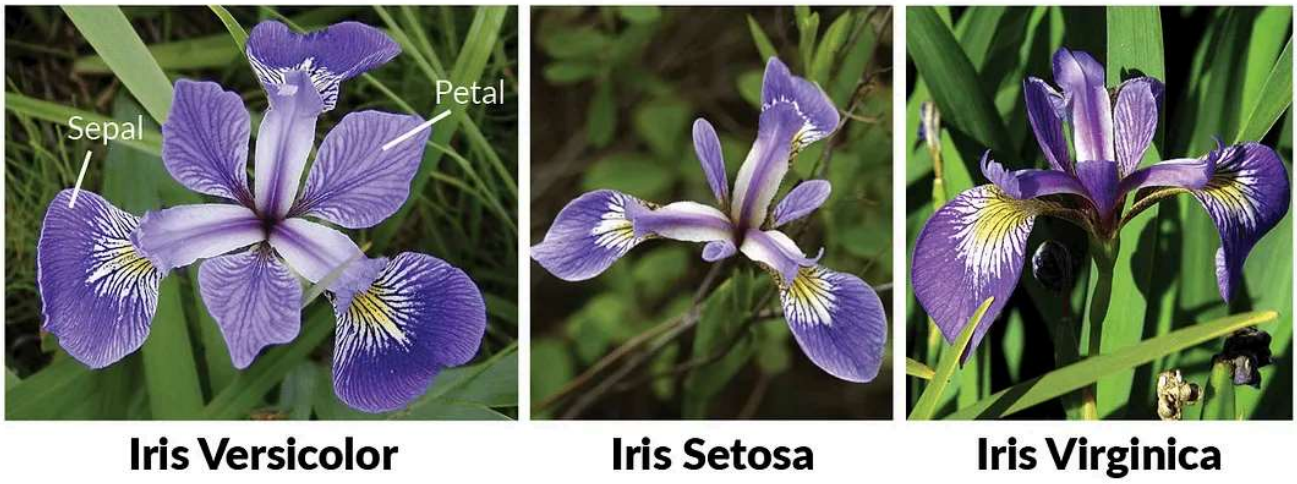
Step-by-step guide on how you can build your first classifier in Python.

Dehao Zhang · Follow
Published in Towards Data Science
12 min read · Jul 13, 2020

Listen Share

Photo by Kevin CASTEL on Unsplash

For a moment, imagine that you are not a flower expert (if you are an expert, good for you!). Can you distinguish between three different species of iris — setosa, versicolor, and virginica?



Credit: Kishan Maladkar (link)

I know I can't...

BUT what if we have a dataset that contains instances of these species, with measurements of their sepals and petals?

In other words, can we **learn** anything from this dataset that would help us distinguish between the three species?

Table of Curiosities

Dataset

In this blog post, I will explore **the Iris dataset** from the UCI Machine Learning Repository. Excerpted from its website, it is said to be "*perhaps **the best known database** to be found in the pattern recognition literature*" [1]. In addition, Jason Brownlee who started the community of Machine Learning Mastery called it the "**Hello World**" of machine learning [2].

I would recommend this dataset to anyone who is a beginner in data science and is eager to build their first ML model. See below for some of nice characteristics of this dataset:

1. 150 samples, with 4 attributes (same units, all numeric)
2. Balanced class distribution (50 samples for each class)
3. No missing data

As you can see, these characteristics can help minimize the time you need to spend in the data preparation process so you can focus on building the ML model. It is NOT that the preparation stage is not important. On the contrary, this process is so important that it can be too time-consuming for some beginners that they may overwhelm themselves before getting to the model development stage.

As an example, the popular dataset House Prices: Advanced Regression Techniques from Kaggle has about 80 features and more than 20% of them contain some level of missing data. In that case, you might need to spend some time understanding the attributes and imputing missing values.

Now hopefully your confidence level (no stats pun intended) is relatively high. Here are some resources on data wrangling that you can read through as you work on more complex datasets and tasks: Dimensionality reduction, Imbalanced classification, Feature engineering, and Imputation.

Objectives

There are two questions that we want to be able to answer after exploring this dataset, which are quite typical in most classification problems:

1. **Prediction** — given new data points, how accurately can the model predict their classes (species)?
2. **Inference** — Which predictor(s) can effectively help with the predictions?

A Few Words on Classification

Classification is a type of **supervised machine learning** problem where the target (response) variable is categorical. Given the training data, which contains the known label, the classifier approximates a mapping function (f) from the input variables (X) to output variables (Y). For more sources on classification, see Chapter 3 in An Introduction to Statistical Learning, Andrew Ng's Machine Learning Course (Week 3), and Simplilearn's tutorial on Classification.

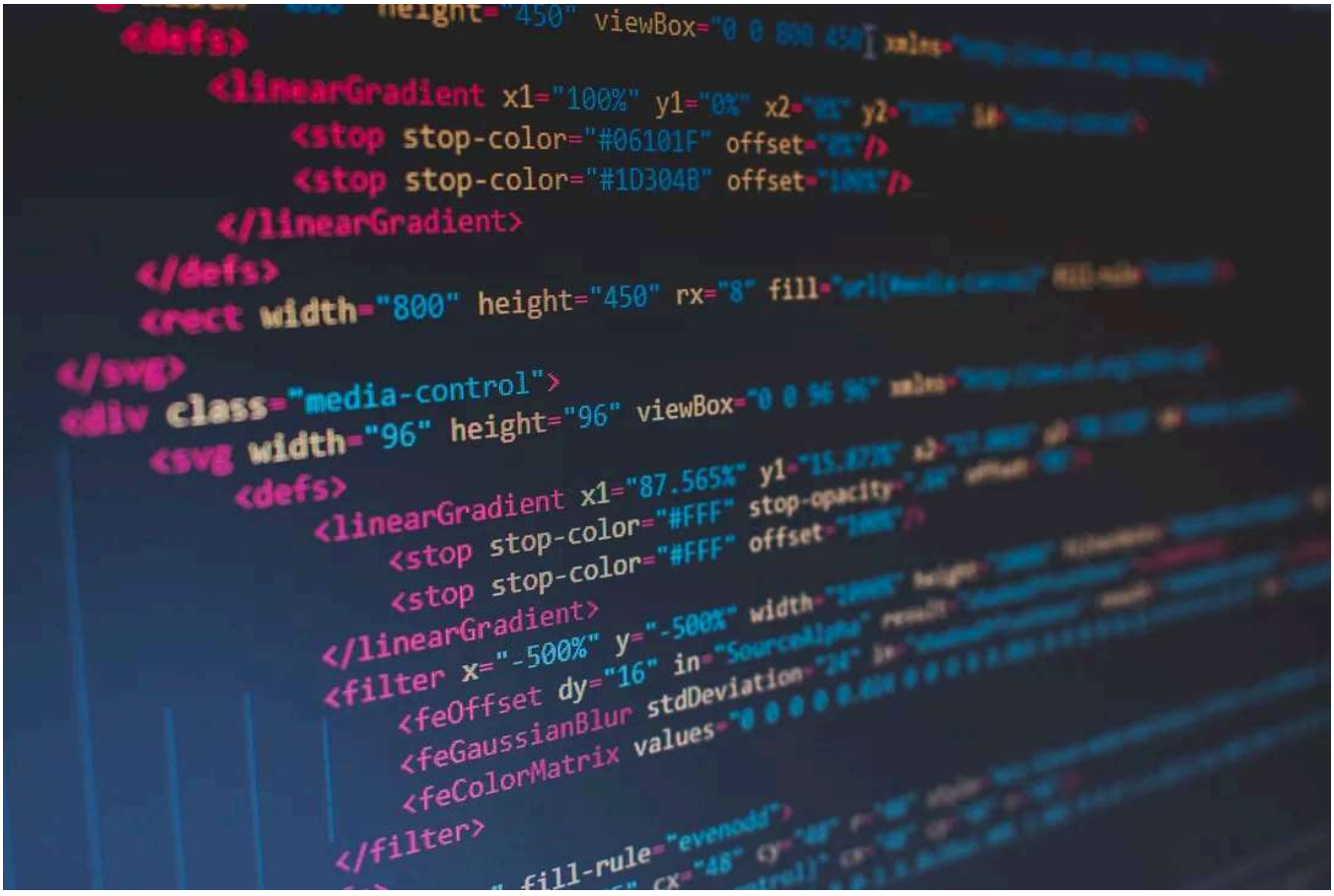


Photo by Florian Olivo on Unsplash

Now it's time to write some code! See my Github page for my full Python code (written in Jupyter Notebook).

Import Libraries and Load Dataset

First, we need to import some libraries: *pandas* (loading dataset), *numpy* (matrix manipulation), *matplotlib* and *seaborn* (visualization), and *sklearn* (building classifiers). Make sure they are installed already before importing them (guide on installing packages here).

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from pandas.plotting import parallel_coordinates
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
```

To load the dataset, we can use the *read_csv* function from pandas (my code also includes the option of loading through url).

```
data = pd.read_csv('data.csv')
```

After we load the data, we can take a look at the first couple of rows through the *head* function:

```
data.head(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Note: all four measurements are in centimeters.

Numerical Summary

First, let's look at a numerical summary of each attribute through *describe:*

```
data.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
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

We can also check the class distribution using *groupby* and *size*:

```
data.groupby('species').size()
```

```
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

We can see that each class has the same number of instances.

Train-Test Split

Now, we can split the dataset into a training set and a test set. In general, we should also have a **validation set**, which is used to evaluate the performance of each classifier and fine-tune the model parameters in order to determine the best model. The test set is mainly used for reporting purposes. However, due to the small size of this dataset, we can simplify this process by using the test set to serve the purpose of the validation set.

In addition, I used a stratified hold-out approach to estimate model accuracy. The other approach is to do cross-validation to reduce bias and variances.

```
train, test = train_test_split(data, test_size = 0.4, stratify =
data['species'], random_state = 42)
```

Note: The general rule of thumb is have 20–30% of dataset as the test set. Due to the small size of this dataset, I chose 40% to ensure there are enough data points to test the model performance.

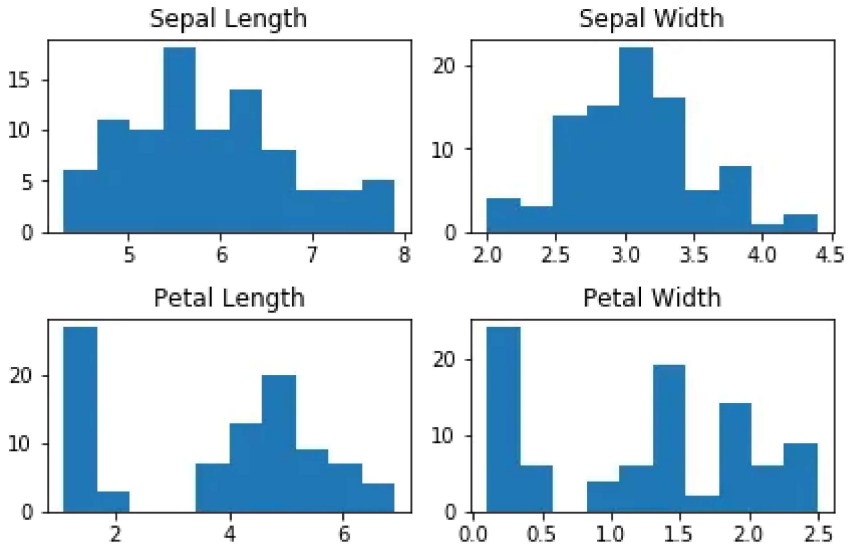
Exploratory Data Analysis

After we split the dataset, we can go ahead to explore the training data. Both *matplotlib* and *seaborn* have great plotting tools then we can use for visualization.

Let's first create some univariate plots, through a histogram for each feature:

```
n_bins = 10
fig, axs = plt.subplots(2, 2)
axs[0,0].hist(train['sepal_length'], bins = n_bins);
axs[0,0].set_title('Sepal Length');
axs[0,1].hist(train['sepal_width'], bins = n_bins);
axs[0,1].set_title('Sepal Width');
axs[1,0].hist(train['petal_length'], bins = n_bins);
axs[1,0].set_title('Petal Length');
axs[1,1].hist(train['petal_width'], bins = n_bins);
axs[1,1].set_title('Petal Width');

# add some spacing between subplots
fig.tight_layout(pad=1.0);
```

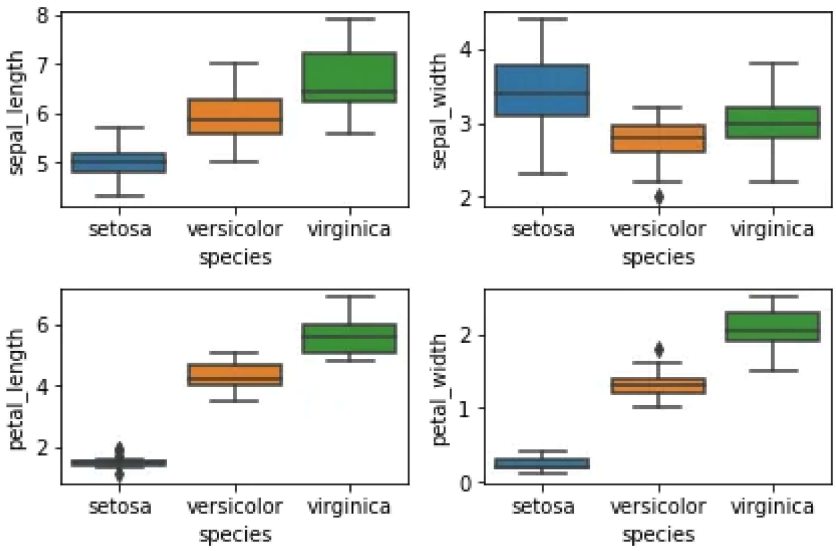


Histograms for the four features

Note that for both petal_length and petal_width, there seems to be a group of data points that have smaller values than the others, suggesting that there might be different groups in this data.

Next, let's try some side-by-side box plots:

```
fig, axs = plt.subplots(2, 2)
fn = ["sepal_length", "sepal_width", "petal_length", "petal_width"]
cn = ['setosa', 'versicolor', 'virginica']
sns.boxplot(x = 'species', y = 'sepal_length', data = train, order =
cn, ax = axs[0,0]);
sns.boxplot(x = 'species', y = 'sepal_width', data = train, order =
cn, ax = axs[0,1]);
sns.boxplot(x = 'species', y = 'petal_length', data = train, order =
cn, ax = axs[1,0]);
sns.boxplot(x = 'species', y = 'petal_width', data = train,  order =
cn, ax = axs[1,1]);
# add some spacing between subplots
fig.tight_layout(pad=1.0);
```

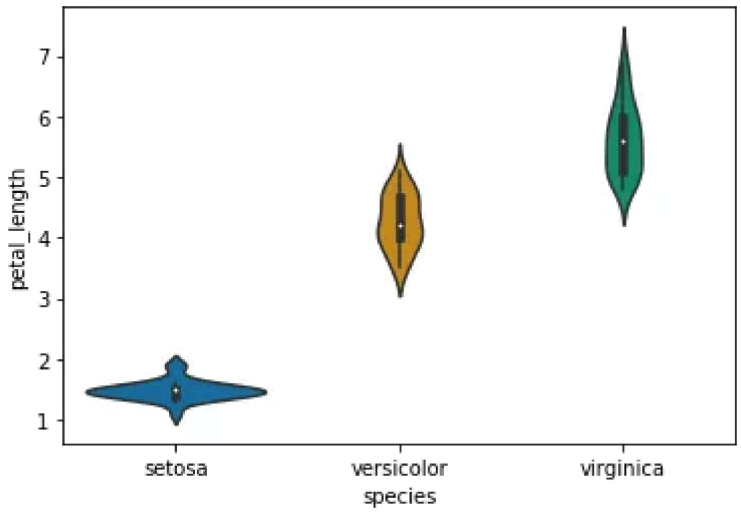


Side-by-side box plots

The two plots at the bottom suggest that that group of data points we saw earlier are **setosas.** Their petal measurements are smaller and less spread-out than those of the other two species as well. Comparing the other two species, versicolor has lower values than virginica on average.

Violin plot is another type of visualization, which combines the benefit of both histogram and box plot:

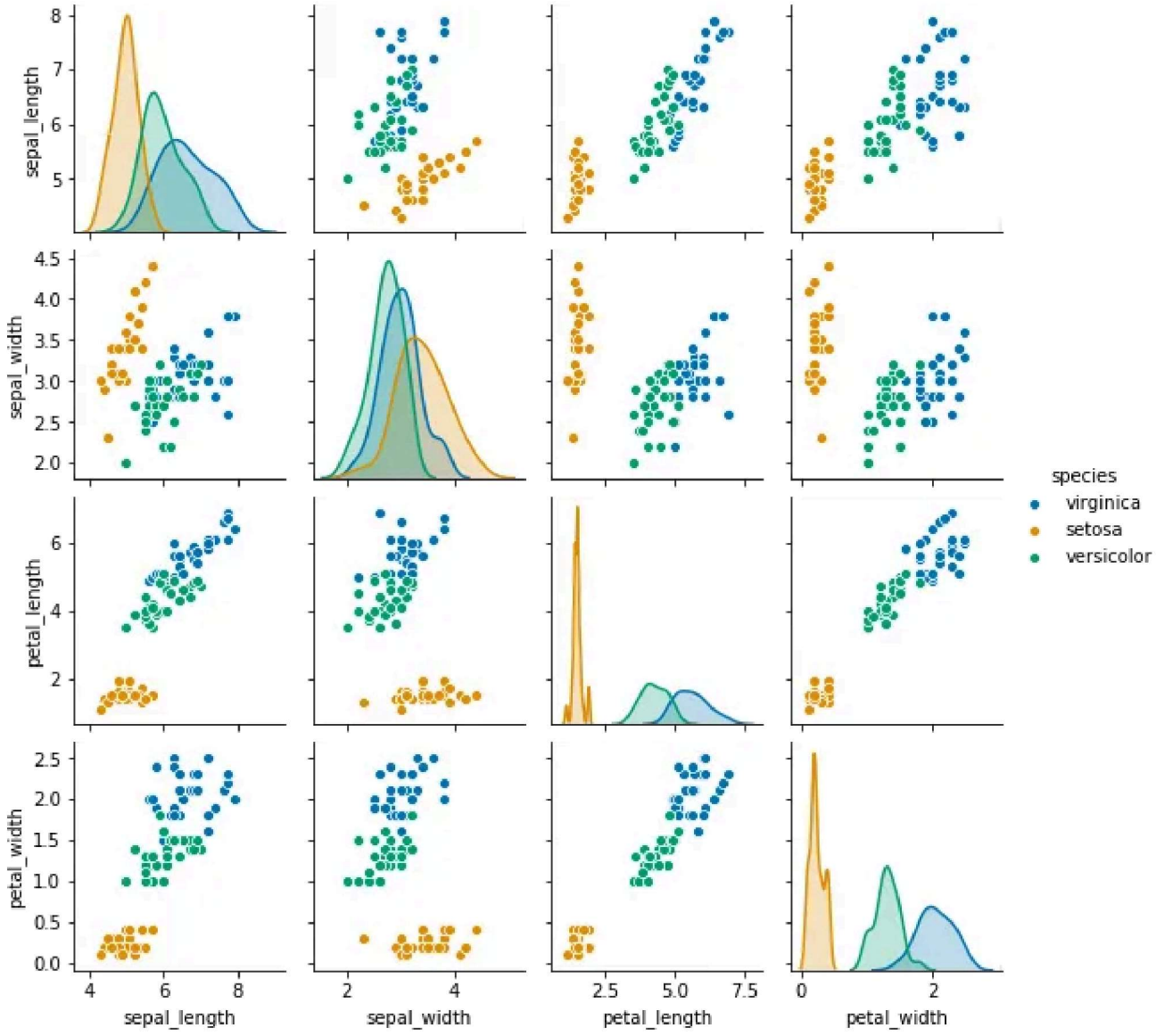
```
sns.violinplot(x="species", y="petal_length", data=train, size=5,
order = cn, palette = 'colorblind');
```



Violin plot for petal_length

Now we can make scatterplots of all-paired attributes by using seaborn's *pairplot* function:

```
sns.pairplot(train, hue="species", height = 2, palette =
'colorblind');
```

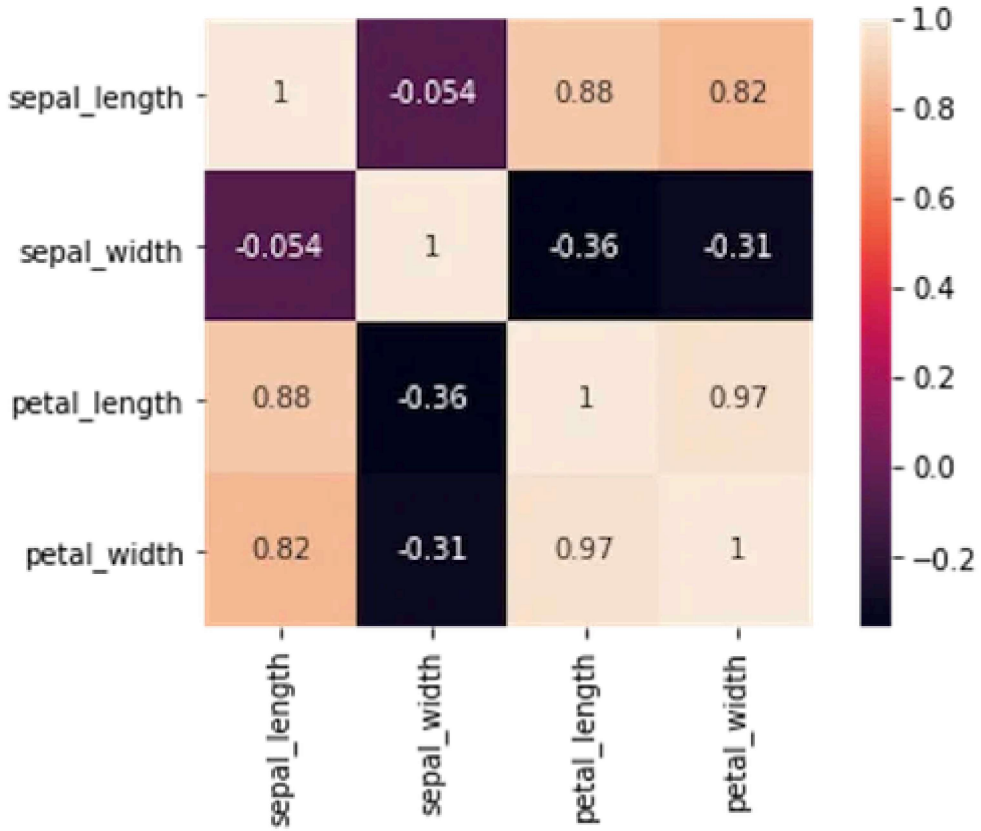


Scatterplots of all-paired attributes

Note that some variables seem to be highly correlated, e.g., petal_length and petal_width. In addition, the petal measurements separate the different species better than the sepal ones.

Next, let's make a correlation matrix to quantitatively examine the relationship between variables:

```
corrmat = train.corr()
sns.heatmap(corrmat, annot = True, square = True);
```

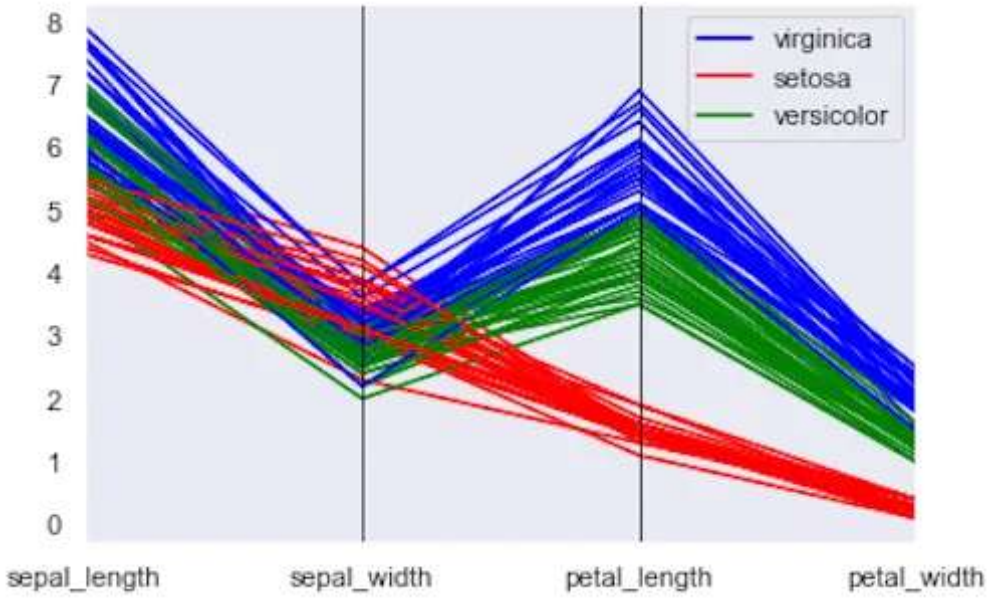


Correlation Matrix

The main takeaway is that the petal measurements have **highly positive correlation**, while the sepal one are uncorrelated. Note that the petal features also have relatively high correlation with sepal_length, but not with sepal_width.

Another cool visualization tool is parallel coordinate plot, which represents each sample as a line.

```
parallel_coordinates(train, "species", color = ['blue', 'red',
'green']);
```



Parallel coordinate plot

As we have seen before, petal measurements can separate species better than the sepal ones.

Build Classifiers

Now we are ready to build some classifiers (woo-hoo!)

To make our lives easier, let's separate out the class label and features first:

```
X_train =
train[['sepal_length','sepal_width','petal_length','petal_width']]
y_train = train.species
X_test =
test[['sepal_length','sepal_width','petal_length','petal_width']]
y_test = test.species
```

Classification Tree

The first classifier that comes up to my mind is a discriminative classification model called classification trees (read more here). The reason is that we get to see the classification rules and it is easy to interpret.

Let's build one using sklearn (documentation), with a maximum depth of 3, and we can check its accuracy on the test data:

```
mod_dt = DecisionTreeClassifier(max_depth = 3, random_state = 1)
mod_dt.fit(X_train,y_train)
prediction=mod_dt.predict(X_test)
print('The accuracy of the Decision Tree
is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))

----------------------------------------------------------------------
The accuracy of the Decision Tree is 0.983.
```

This decision tree predicts 98.3% of the test data correctly. One nice thing about this model is that you can see the importance of each predictor through its *feature_importances_* attribute:

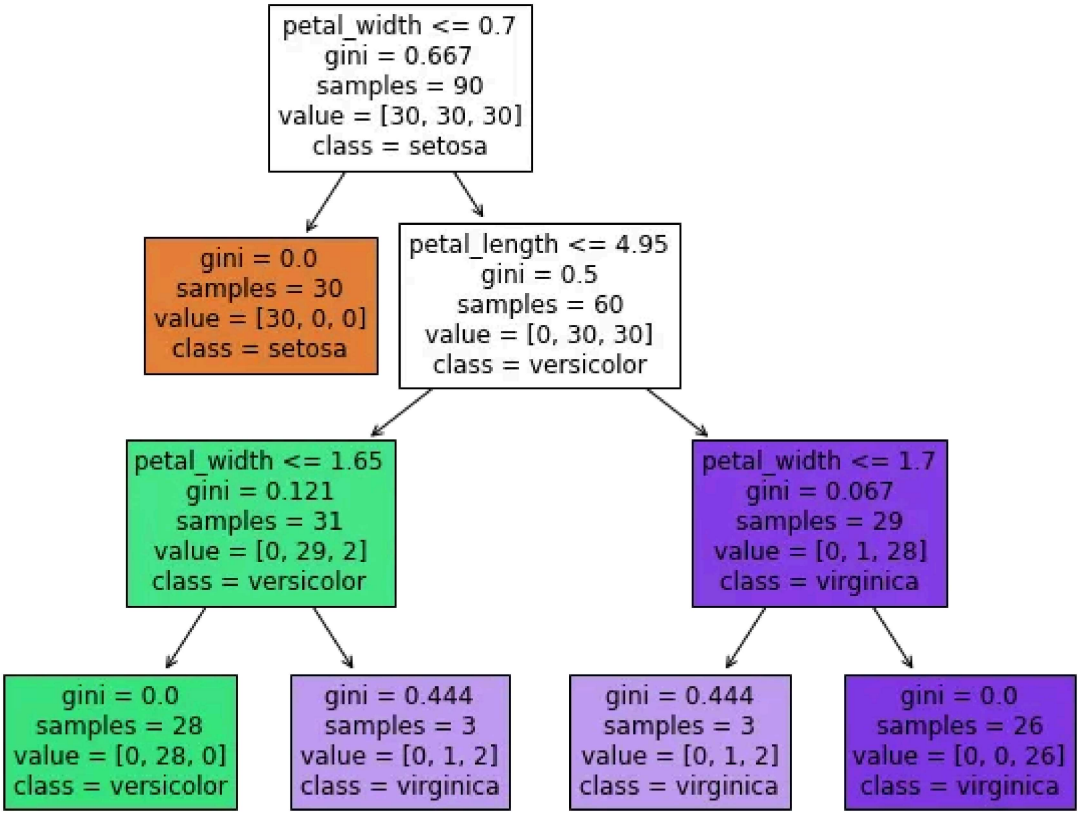
```
mod_dt.feature_importances_

----------------------------------------------------------------------
array([0.        , 0.        , 0.42430866, 0.57569134])
```

From the output and based on the indices of the four features, we know that the first two features (sepal measurements) are of no importance, and only the petal ones are used to build this tree.

Another nice thing about the decision tree is that we can visualize the classification rules through *plot_tree*:

```
plt.figure(figsize = (10,8))
plot_tree(mod_dt, feature_names = fn, class_names = cn, filled =
True);
```

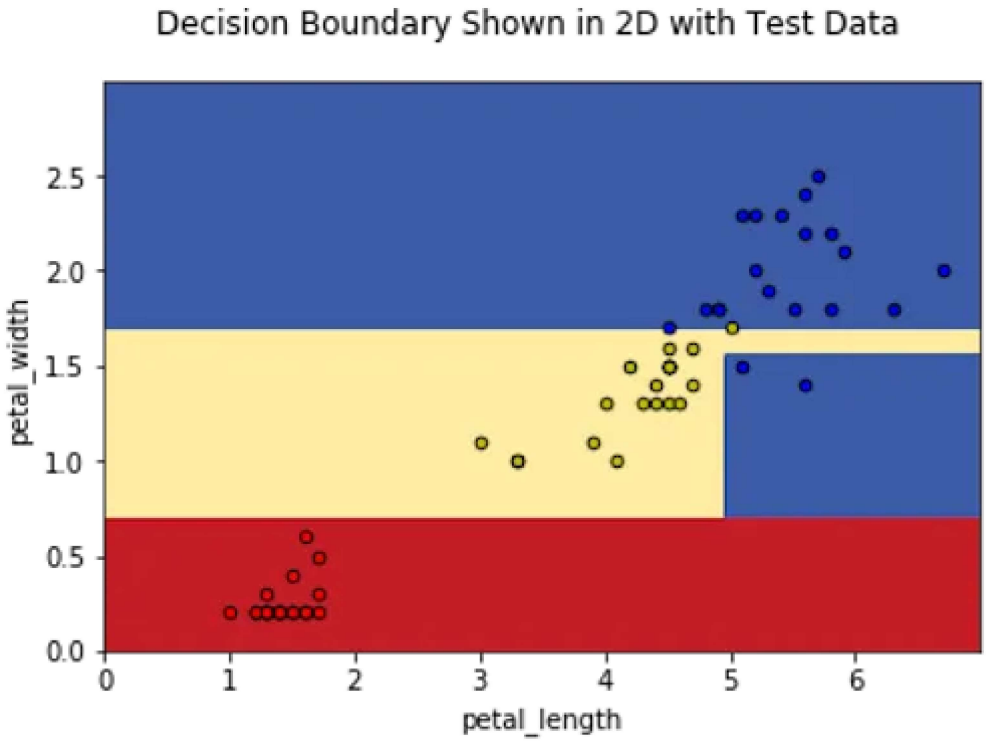


Classification rules from this tree (for each split, left ->yes, right ->no)

Apart from each rule (e.g. the first criterion is petal_width ≤ 0.7), we can also see the Gini index (impurity measure) at each split, assigned class, etc. Note that all terminal nodes are pure besides the two “light purple” boxes at the bottom. We can less confident regarding instances in those two categories.

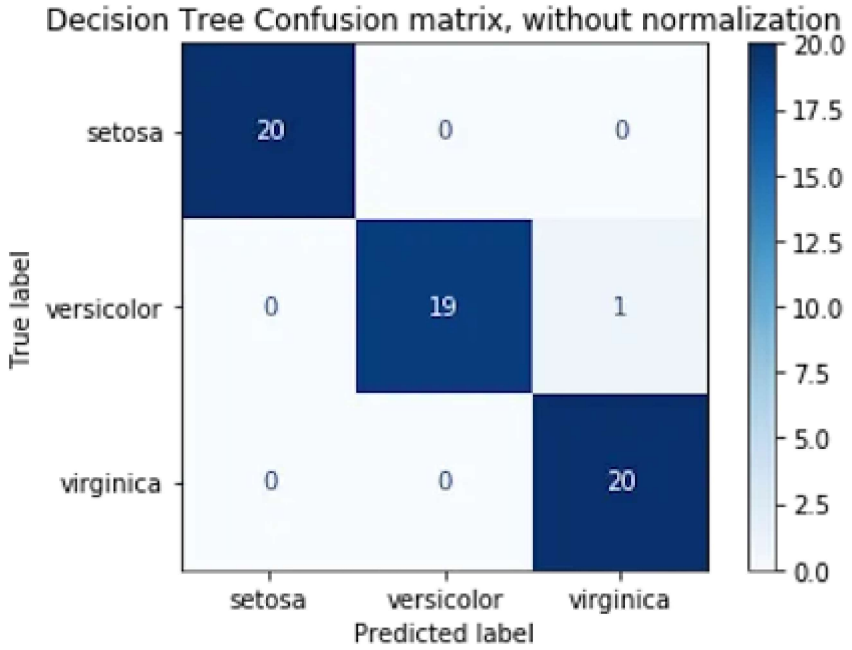
To demonstrate how easy it is to classify new data points, say a new instance has a petal length of 4.5cm and a petal width of 1.5cm, then we can predict it to be versicolor following the rules.

Since only the petal features are being used, we can visualize the decision boundary and plot the test data in 2D:



Out of the 60 data points, 59 are correctly classified. Another way to show the prediction results is through a confusion matrix:

```
disp = metrics.plot_confusion_matrix(mod_dt, X_test, y_test,
                                     display_labels=cn,
                                     cmap=plt.cm.Blues,
                                     normalize=None)
disp.ax_.set_title('Decision Tree Confusion matrix, without
normalization');
```



Through this matrix, we see that there is one versicolor which we predict to be virginica.

One downside is building a single tree is its **instability**, which can be improved through ensemble techniques such as random forests, boosting, etc. For now, let's move on to the next model.

Gaussian Naive Bayes Classifier

One of the most popular classification models is Naive Bayes. It contains the word "Naive" because it has a key assumption of **class-conditional independence**, which means that given the class, each feature's value is assumed to be independent of that of any other feature (read more here).

We know that it is clearly not the case here, evidenced by the high correlation between the petal features. Let's examine the test accuracy using this model to see whether this assumption is robust:

```
The accuracy of the Guassian Naive Bayes Classifier on test data is
0.933
```

What about the result if we only use the petal features:

```
The accuracy of the Guassian Naive Bayes Classifier with 2
predictors on test data is 0.950
```

Interestingly, using only two features results in more correctly classified points, suggesting possibility of over-fitting when using all features. Seems that our Naive Bayes classifier did a decent job.

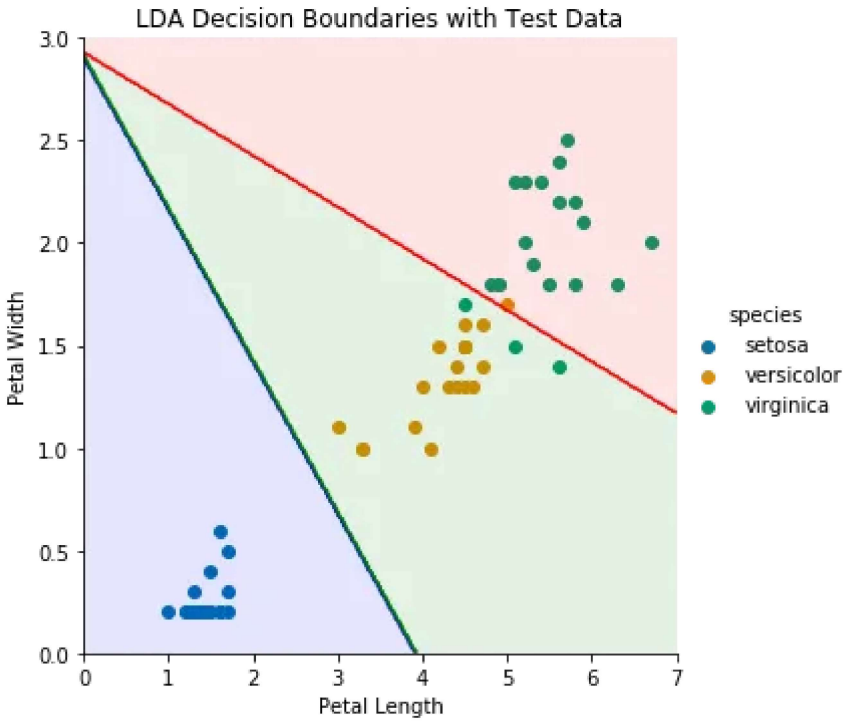
Linear Discriminant Analysis (LDA)

If we use **multivariate Gaussian distribution** to calculate the class conditional density instead of taking a product of univariate Gaussian distribution (used in Naive Bayes), we would then get a LDA model (read more here). The key assumption of LDA is that the covariances are equal among classes. We can examine the test accuracy using all features and only petal features:

```
The accuracy of the LDA Classifier on test data is 0.983
The accuracy of the LDA Classifier with two predictors on test data
is 0.933
```

Using all features boosts the test accuracy of our LDA model.

To visualize the decision boundary in 2D, we can use our LDA model with only petals and also plot the test data:



Four test points are misclassified — three virginica and one versicolor.

Now suppose we want to classify new data points with this model, we can just plot the point on this graph, and predicts according to the colored region it belonged to.

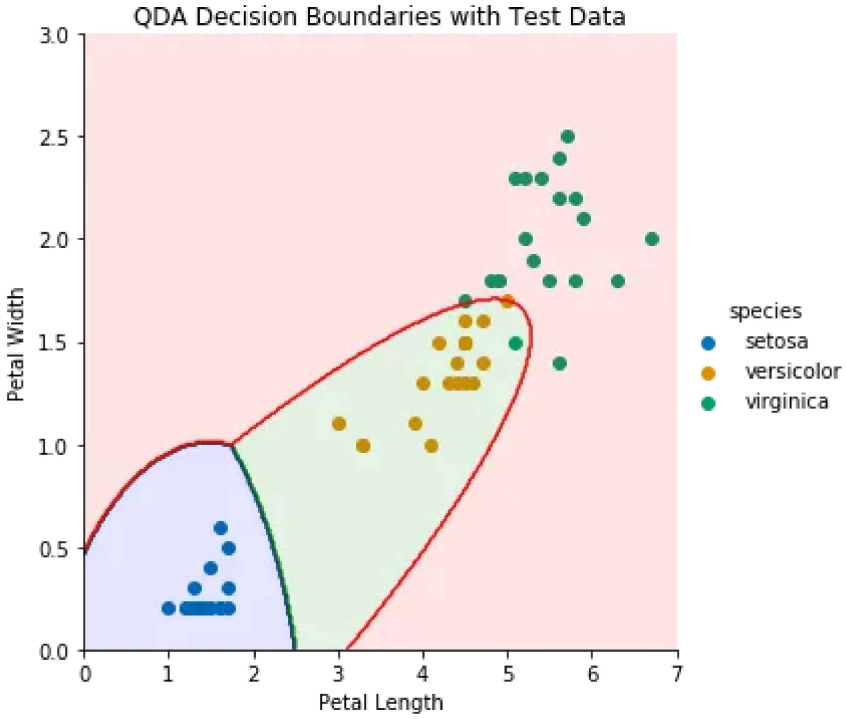
Quadratic Discriminant Analysis (QDA)

The difference between LDA and QDA is that QDA does NOT assume the covariances to be equal across classes, and it is called “quadratic” because the decision boundary is a quadratic function.

```
The accuracy of the QDA Classifier is 0.983
The accuracy of the QDA Classifier with two predictors is 0.967
```

It has the same accuracy with LDA in the case of all features, and it performs slightly better when only using petals.

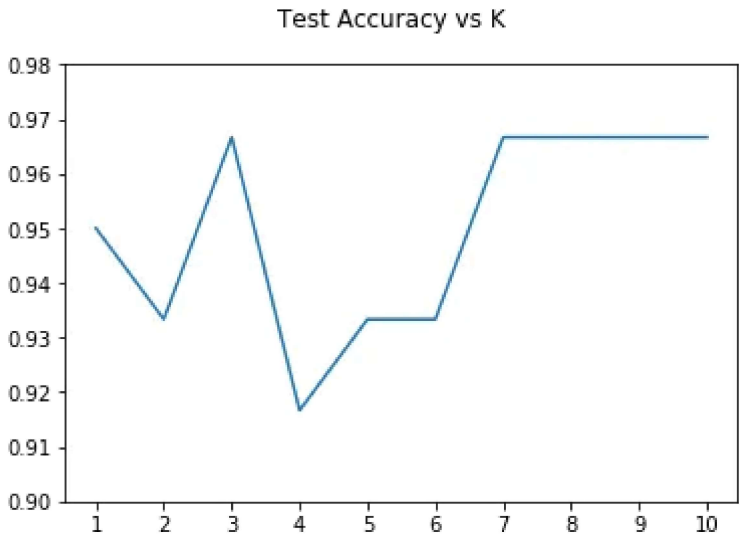
Similarly, let's plot the decision boundary for QDA (model with only petals):



K Nearest Neighbors (K-NN)

when number of features gets large.

Let's plot the test accuracy with different choices of K:



We can see that the accuracy is highest (about 0.965) when K is 3, or between 7 and 10. Compare to the previous models, it is less straightforward to classify new data points since we would need to look at its K closest neighbors in four-dimensional space.

Other Models

I also explored other models such as logistic regression, support vector machine classifier, etc. See my code on Github for details.

Note that the **SVC (with linear kernel)** achieved a test accuracy of 100%!

We should be pretty confident now since most of our models performed better than 95% accuracy.

Next Steps

Here are a few ideas for future studies:

1. Create a validation set and run cross-validation to get accurate estimates and compare the spread and mean accuracy between each of them.
2. Find other data sources that include other iris species and their sepal/petal measurements (include other attributes too if possible), and examine the new classification accuracy.
3. Make an interactive web app that predicts species given measurements inputted from users (Check out my simple web demo with Heroku deployment here)

Summary

Let's recap.

We explored the Iris dataset, and then built a few popular classifiers using *sklearn*. We saw that the petal measurements are more helpful at classifying instances than the sepal ones. Furthermore, most models achieved a test accuracy of over 95%.

I hope you enjoy this blog post and please share any thought that you may have :)

Check out my other post on exploring the Yelp dataset:

Discover Your Next Favorite Restaurant — Exploration and Visualization on Yelp Dataset

Do you use Yelp to find good restaurants? This post reveals insights and patterns in the popular Yelp Dataset.

towardsdatascience.com

References

[1] https://archive.ics.uci.edu/ml/datasets/iris

[2] https://machinelearningmastery.com/machine-learning-in-python-step-by-step/

Data Science | Machine Learning | Python | Exploratory Data Analysis | Classification

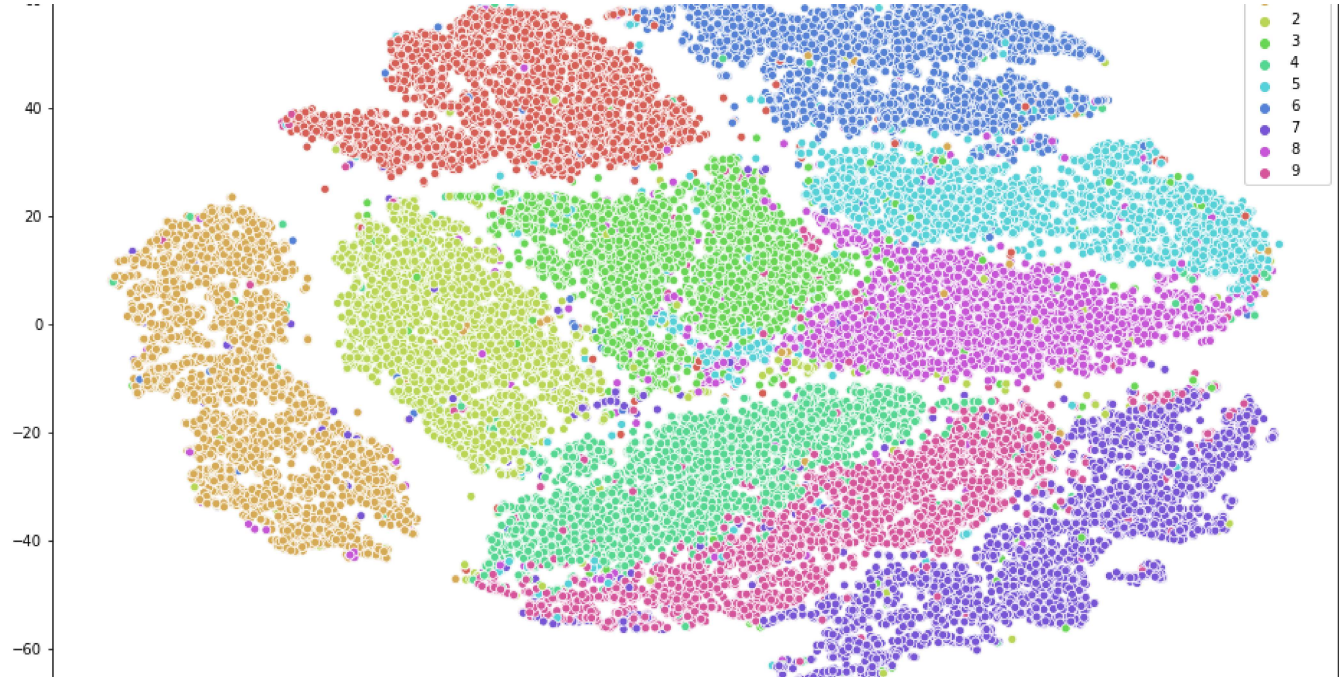
Follow

Written by Dehao Zhang

55 Followers · Writer for Towards Data Science

Data Science | Workplace Analytics | Operations Research | Building Intelligent Solutions

More from Dehao Zhang and Towards Data Science



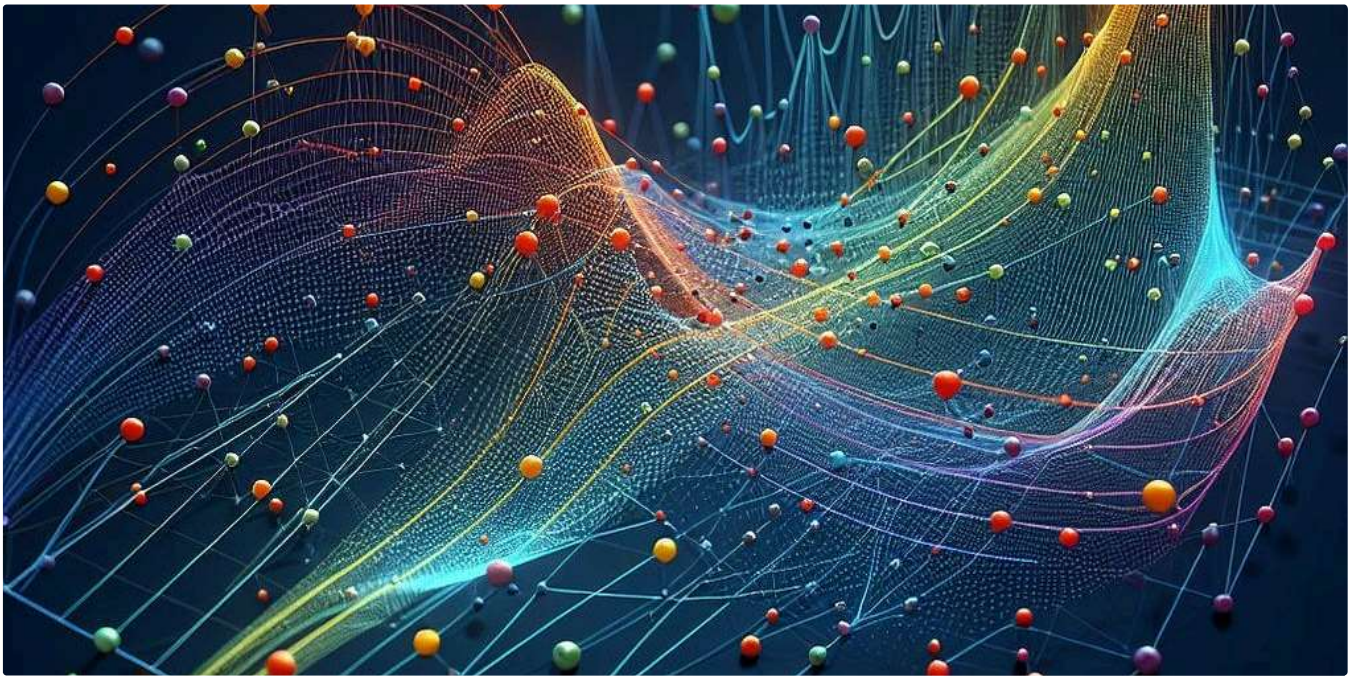
Dehao Zhang in Towards Data Science

Dimensionality Reduction using t-Distributed Stochastic Neighbor Embedding (t-SNE) on the MNIST...

t-SNE vs PCA vs PCA & t-SNE

6 min read · Aug 16, 2020

123 1



Tim Sumner in Towards Data Science

A New Coefficient of Correlation

What if you were told there exists a new way to measure the relationship between two variables just like correlation except possibly...

10 min read · Mar 31, 2024

3K 36

Cristian Leo in Towards Data Science

The Math Behind Neural Networks

Dive into Neural Networks, the backbone of modern AI, understand its mathematics, implement it from scratch, and explore its applications

· 28 min read · Mar 29, 2024

3K 20

Dehao Zhang in Towards Data Science

Create an Image Classification Web App using PyTorch and Streamlit

Build a simple image classification app with 50 lines of code!

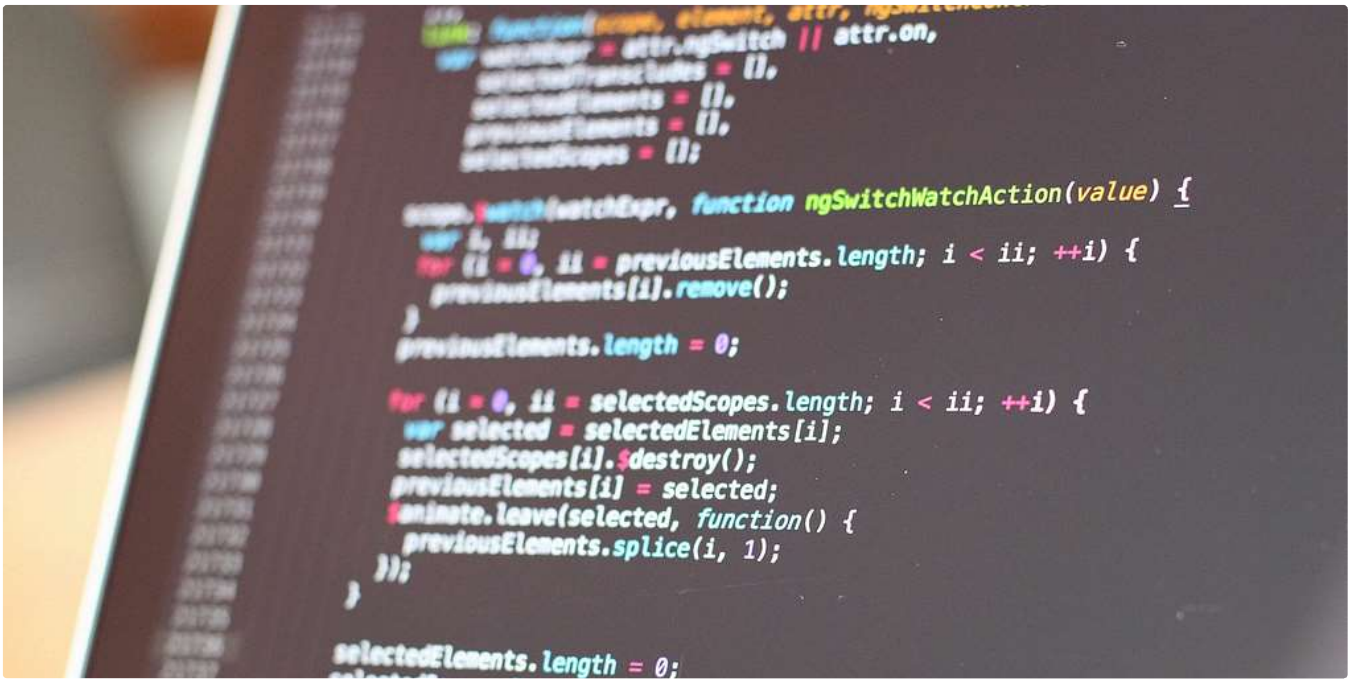
4 min read · Aug 2, 2020

20 1

See all from Dehao Zhang

See all from Towards Data Science

Recommended from Medium



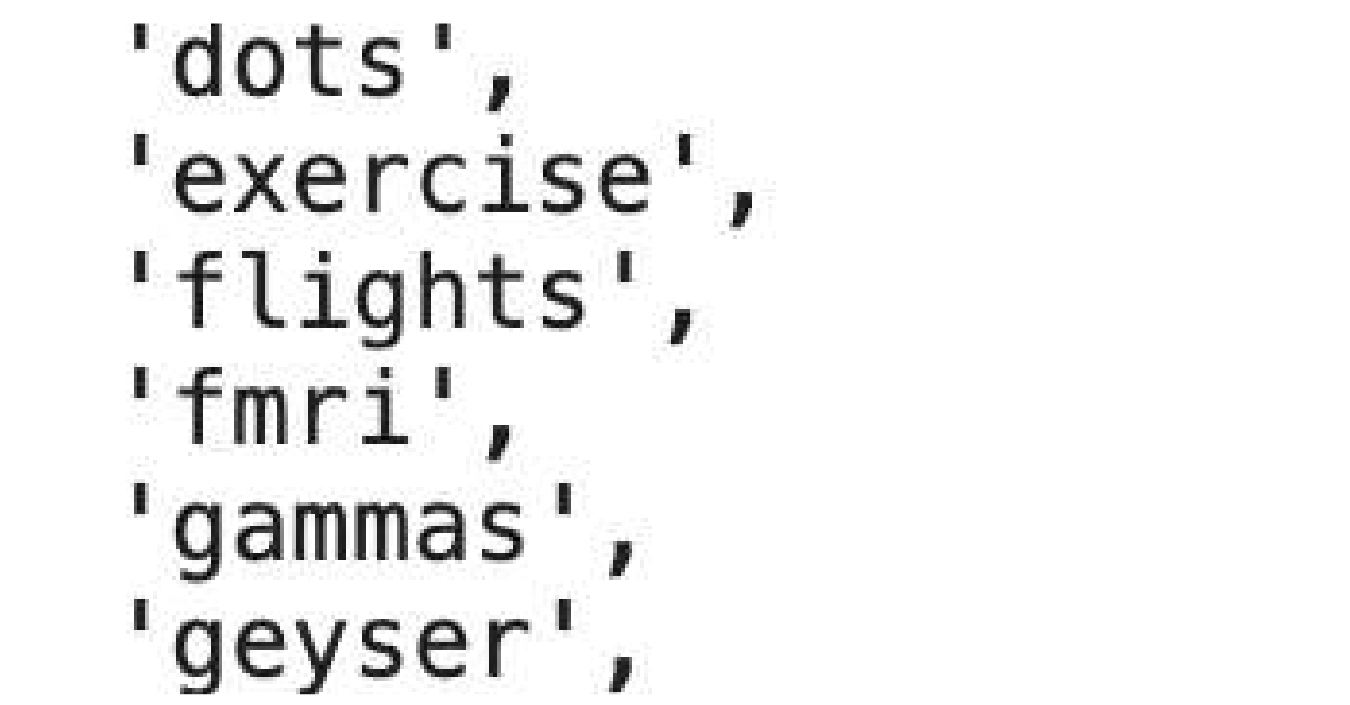
Maxim Gusarov in CodeX

Do I need to tune logistic regression hyperparameters?

Aren't we over-committed to optimizing the data science work we do? We are often trying to find the best combination of x, y, z variables...

9 min read · Apr 9, 2022

58 3



Coursesteach

Supervised learning with scikit-learn (Part 6)-Importing the DataSet

Chapter:3-Data Preprocessing

11 min read · Nov 16, 2023

4

Lists

Predictive Modeling w/ Python

20 stories · 1122 saves

Practical Guides to Machine Learning

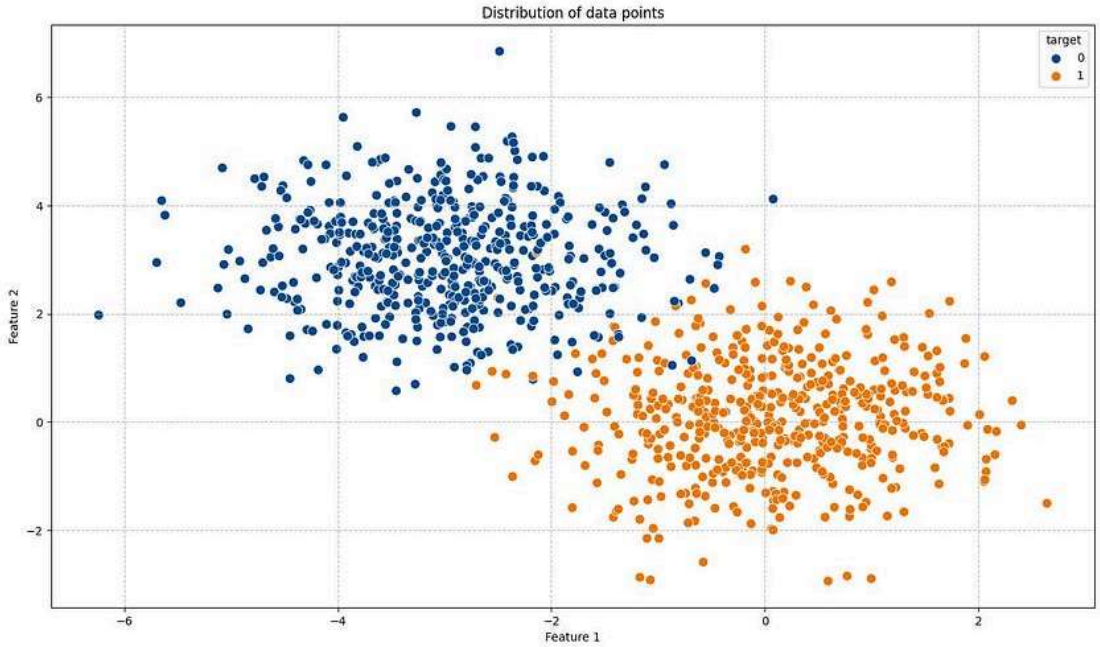
10 stories · 1345 saves

Coding & Development

11 stories · 574 saves

Natural Language Processing

1396 stories · 895 saves



Madhuri Patil

KNN Classifier Implementation: Best Practices and Tips (PART I)

KNN or k-nearest neighbors is a simple, yet powerful machine learning algorithm used for classification and regression tasks.

7 min read · Nov 18, 2023

423	403	20	20	400
345	325	20	20	400
422	400	22	22	484
345	322	23	23	529
367	387	-20	20	400
285	300	-15	15	225
440	460	-20	20	400
		Mean	19.27272727	382.1818182
		Root Mean		19.54947105

Chaitanya (Chey) Penmetsa in CodeNx

Machine learning using Scikit-Learn (sklearn)—Evaluating Regression model using metrics

Scikit-learn, commonly known as sklearn, stands out as one of the most influential and extensively utilized machine learning libraries in...

7 min read · Jan 1, 2024

Sebastien Poletto

Automating Email Classification with AI: Enhance Efficiency

Email management can often feel like an endless battle against a rising tide of incoming messages.

13 min read · Feb 23, 2024

Gen. David L.

Pandas Data Preprocessing — Handling Missing Values

Data preprocessing typically involves the following aspects:

7 min read · Nov 29, 2023