

# AASD 4000

# Machine Learning - I

Applied AI Solutions Developer Program




# Module 9

# ML Algorithms II

Vejeý Gandyer



# Agenda



Model Building Template  
Naïve Bayes  
PCA  
Gaussian Mixture Models  
Benchmarking Algorithms

# ML Algorithm

What is it?



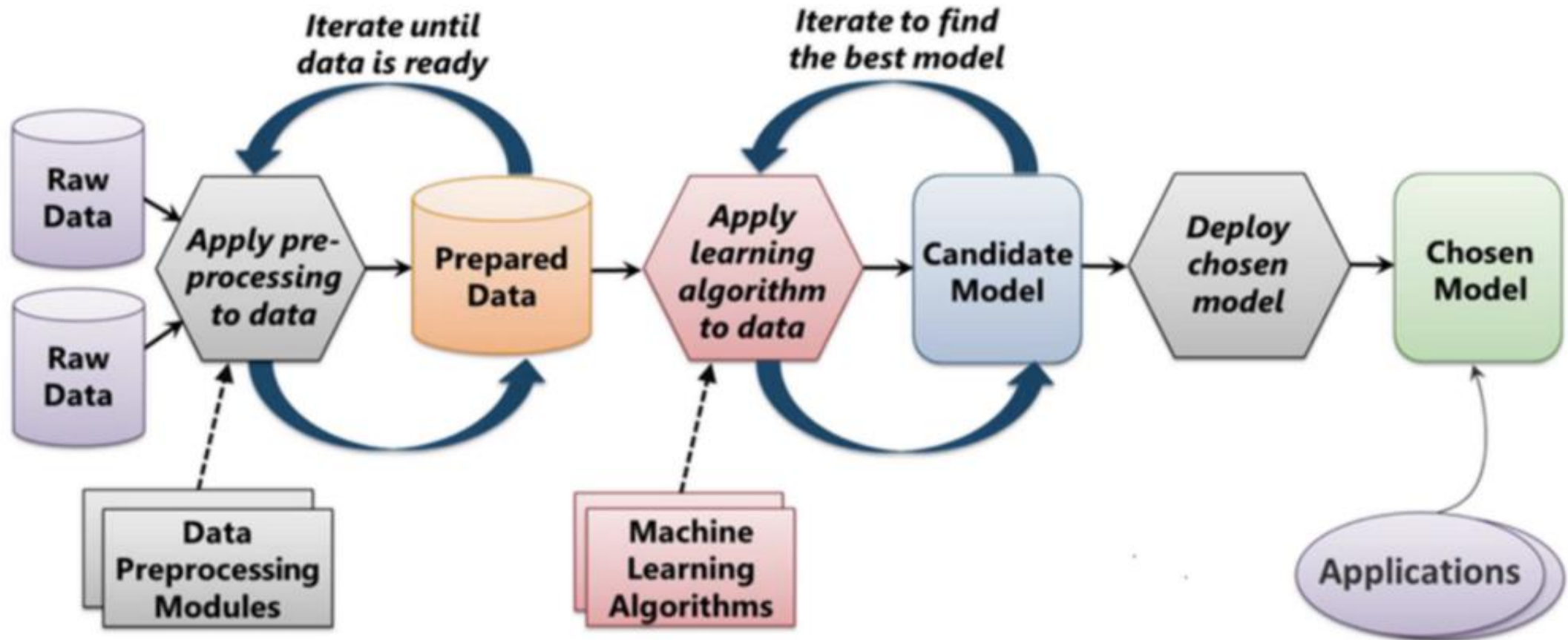
# What is ML Algorithm ?

ML Algorithm is a series of steps that is used to learn a mapping function that converts raw data into set of rules.

Top ML Algorithms you should be familiar with

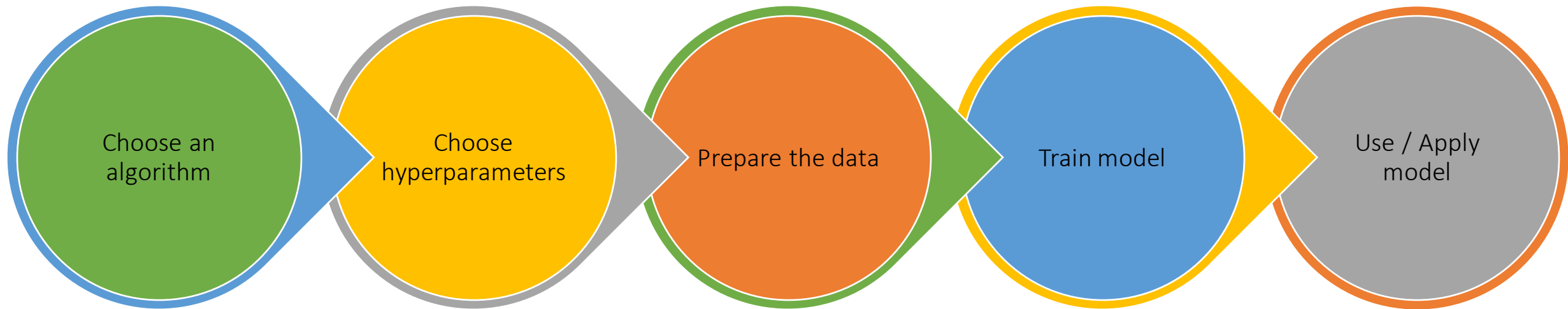
- Linear Regression
- Logistic Regression
- Naïve Bayes
- K-Nearest Neighbours
- Support Vector Machine
- Decision Tree
- Random Forest
- Principal Component Analysis
- K-Means
- XGBoost
- LightGBM

# Machine Learning Process

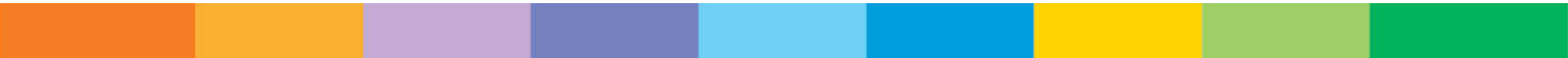


From "Introduction to Microsoft Azure" by David Chappell

# Model Building Template



# Naïve Bayes





# Naïve Bayes

Extremely fast

Very few tunable parameters

Quick-and-dirty baseline algorithm for any classification problem

Generative Classification

Bayes Theorem

$$P(L \mid \text{features}) = \frac{P(\text{features} \mid L)P(L)}{P(\text{features})}$$

$$\frac{P(L_1 \mid \text{features})}{P(L_2 \mid \text{features})} = \frac{P(\text{features} \mid L_1) P(L_1)}{P(\text{features} \mid L_2) P(L_2)}$$

# Naïve Bayes

If we make very **naïve** assumptions about the generative model for each label, we can find a rough approximation of the generative model for each class, and then proceed with the Bayesian classification

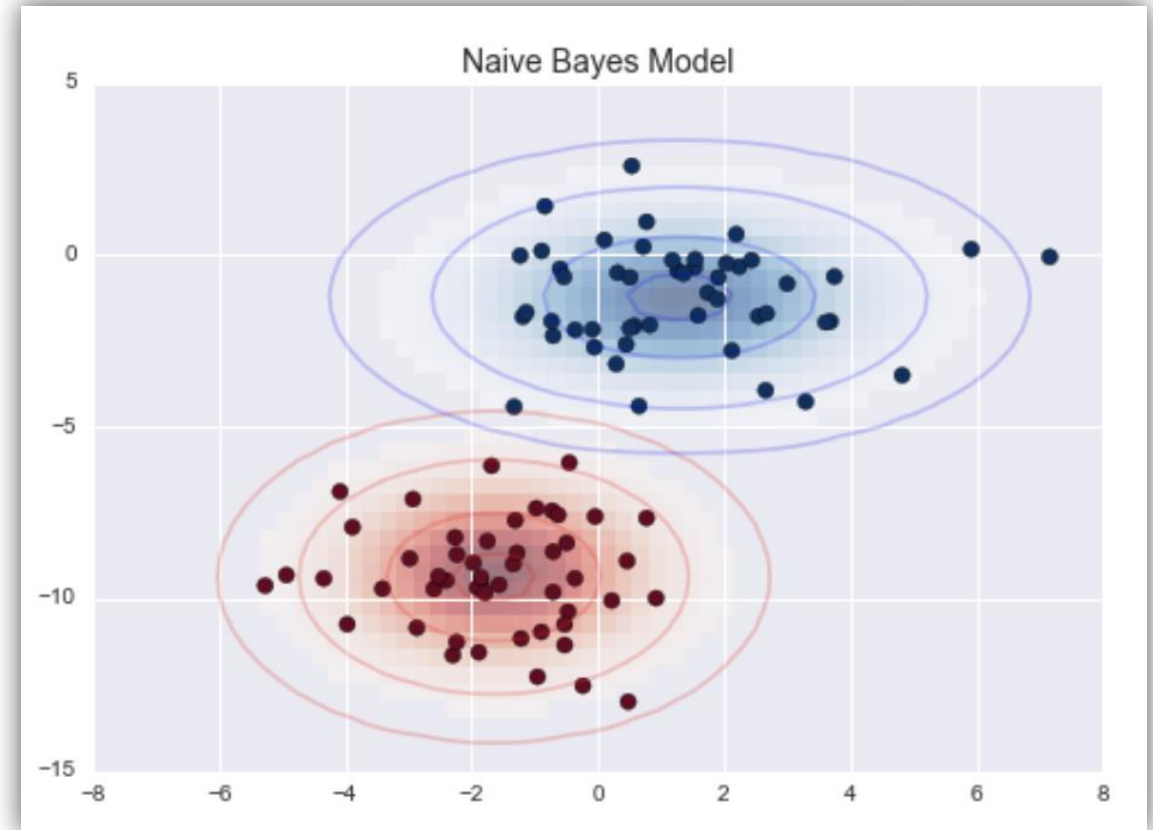
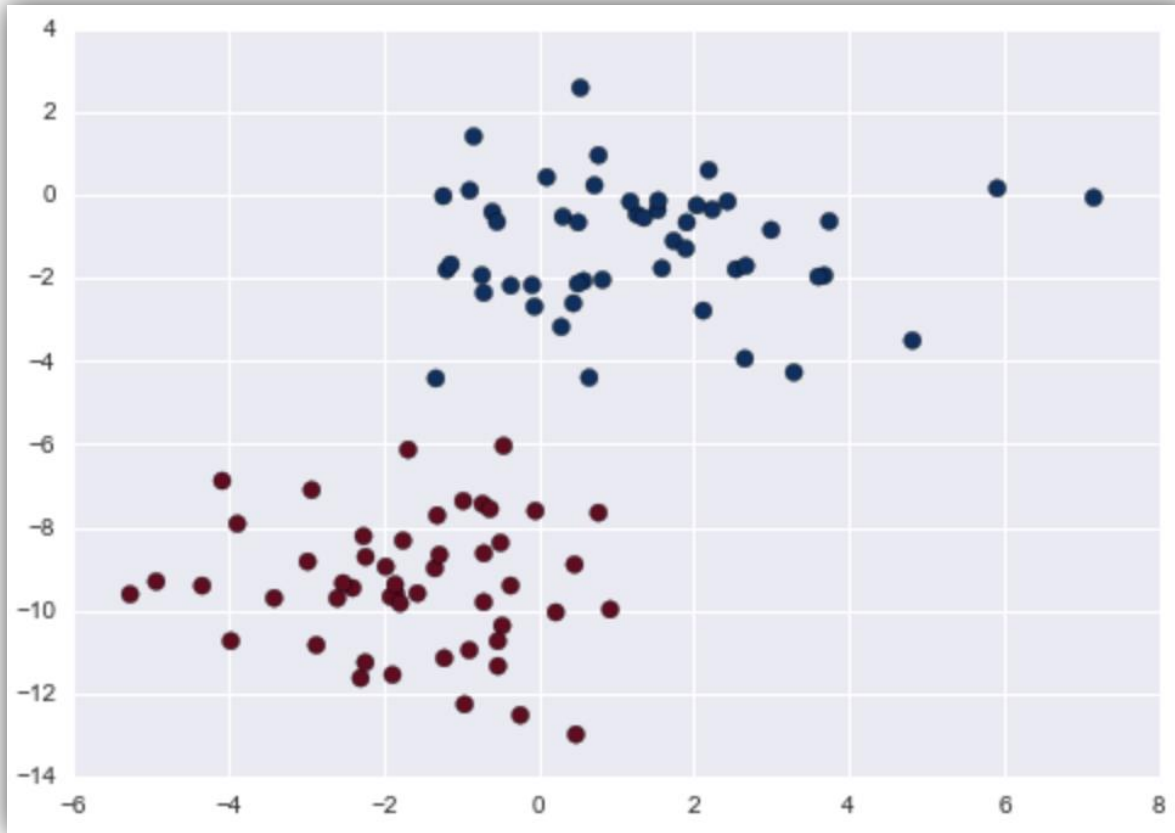
Gaussian Naïve Bayes – assumption is that data from each label is drawn from a simple Gaussian distribution

Multinomial Naïve Bayes - assumption is that data from each label is drawn from a simple Multinomial distribution

# Gaussian Naïve Bayes

Assumption: Data from each label is drawn from a simple Gaussian distribution

```
from sklearn.datasets import make_blobs
X, y = make_blobs(100, 2, centers=2, random_state=2, cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu');
```



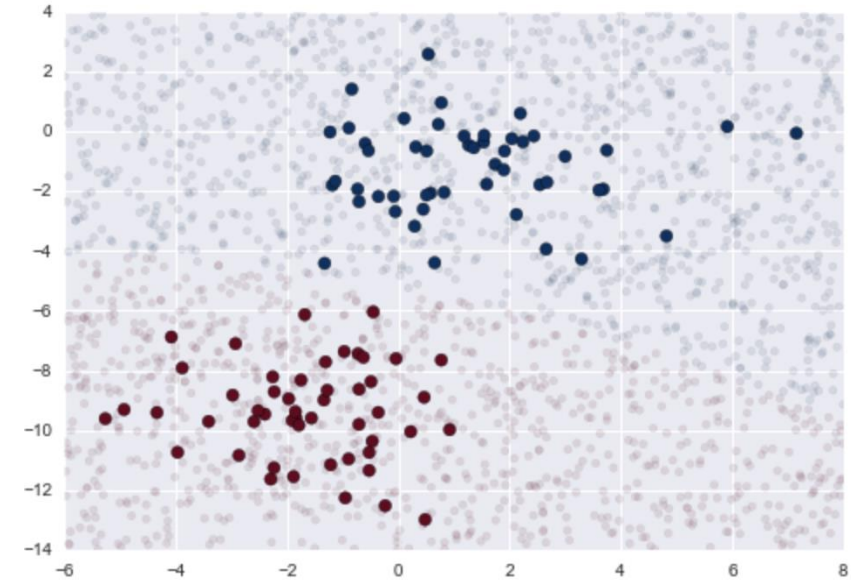
# Gaussian Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB  
model = GaussianNB()  
model.fit(X, y);
```

```
rng = np.random.RandomState(0)  
Xnew = [-6, -14] + [14, 18] * rng.rand(2000, 2)  
ynew = model.predict(Xnew)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu')  
lim = plt.axis()  
plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='RdBu', alpha=0.1)  
plt.axis(lim);
```

```
yprob = model.predict_proba(Xnew)  
yprob[-8:].round(2)
```




```
array([[ 0.89,  0.11],  
       [ 1.   ,  0.   ],  
       [ 1.   ,  0.   ],  
       [ 1.   ,  0.   ],  
       [ 1.   ,  0.   ],  
       [ 1.   ,  0.   ],  
       [ 0.   ,  1.   ],  
       [ 0.15,  0.85]])
```

# Naïve Bayes

## Pros

- They are extremely fast for both training and prediction
- They provide straightforward probabilistic prediction
- They are often very easily interpretable
- They have very few (if any) tunable parameters

## They perform better

- When the naive assumptions actually matches the data
  - For very well-separated categories, when model complexity is less important
  - For very high-dimensional data, when model complexity is less important
- 
- A decorative horizontal bar at the bottom of the slide, composed of several colored rectangular segments: orange, yellow, light purple, dark purple, light blue, dark blue, yellow, light green, and dark green.

# Principal Component Analysis (PCA)



# Principal Component Analysis (PCA)

Unsupervised algorithm

Dimensionality Reduction

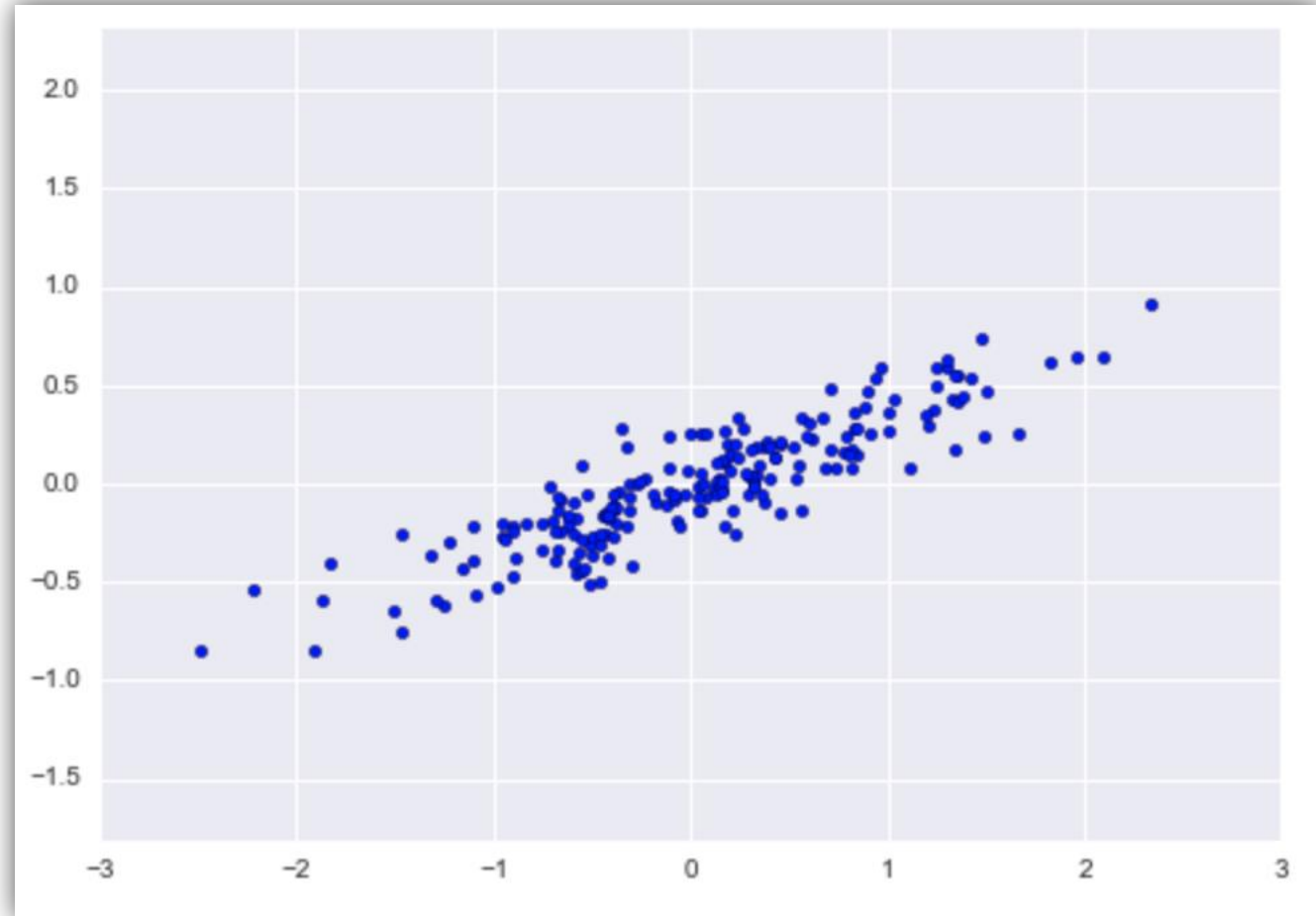
Uses

- Visualization tool
- Noise filter
- Feature Extractor

# Principal Component Analysis (PCA)

Attempts to learn the relationship between  $X$  and  $y$  values

Relationship is defined by finding a list of **principal axes** in the data



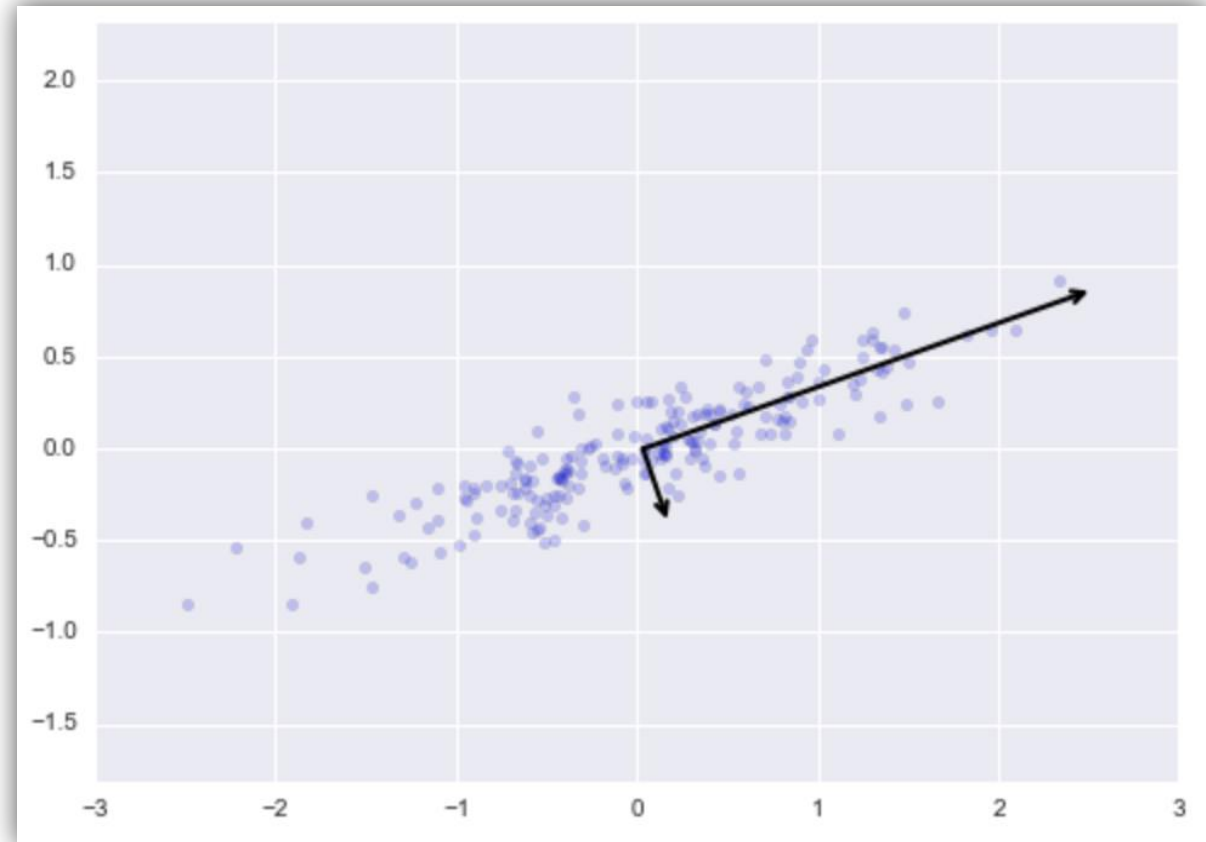


# Principal Component Analysis (PCA)

```
from sklearn.decomposition import PCA  
pca = PCA(n_components=2)  
pca.fit(X)
```

```
print(pca.components_)
```

```
print(pca.explained_variance_)
```

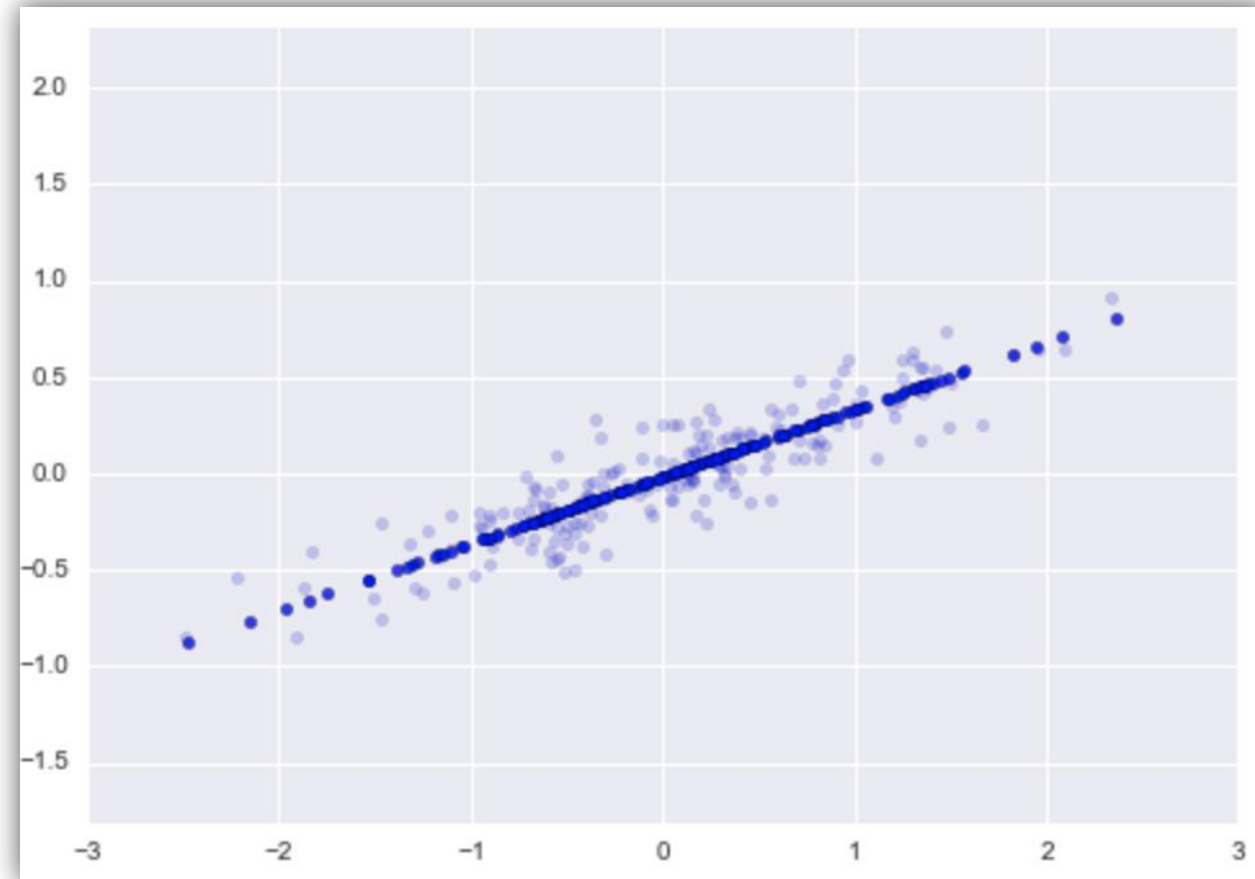


# PCA as Dimensionality Reduction

Zeroing out one or more of the smallest principal components resulting in a low-dimension projection of data that preserves the maximal data variance

```

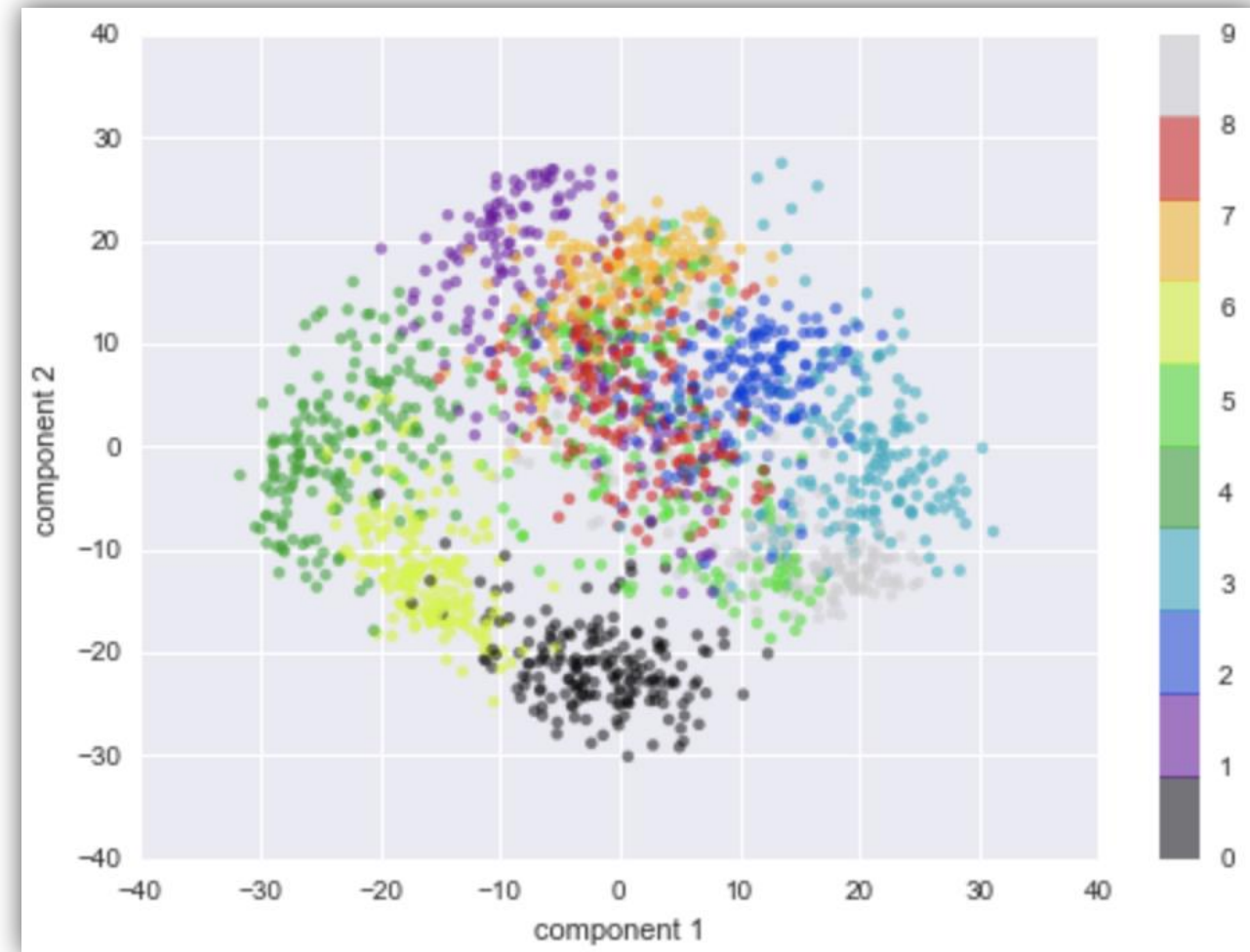
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
    
```



# PCA as Visualization tool

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

```
# project from 64 to 2 dimensions
pca = PCA(2)
reduced = pca.fit_transform(digits.data)
print(digits.data.shape)
print(reduced.shape)
```



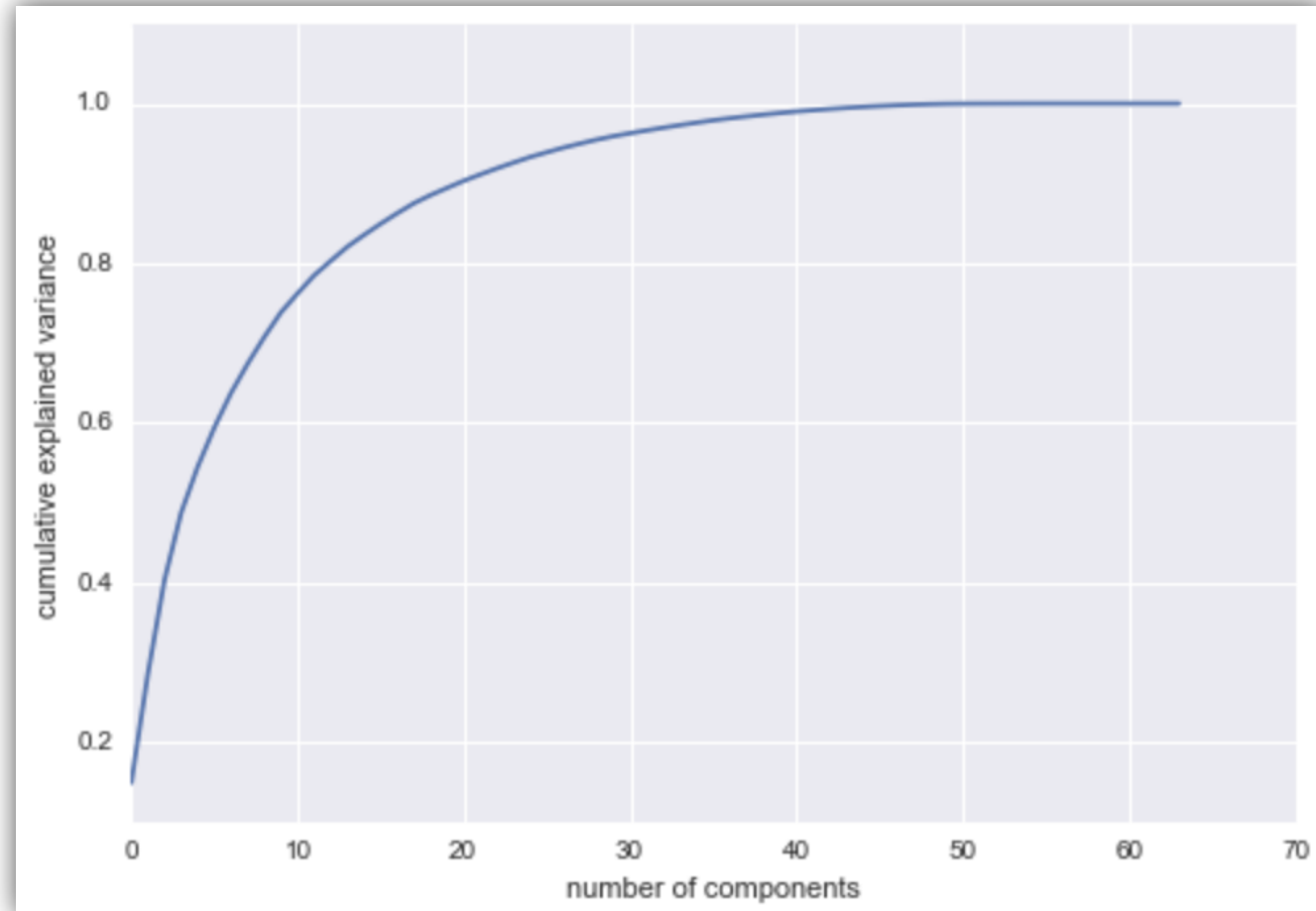
# Choosing the number of components

```
pca = PCA().fit(digits.data)  
plt.plot(np.cumsum(pca.explained_variance_ratio_))
```

10 components  
contains 75% of  
variance

50 components contain  
100% of variance

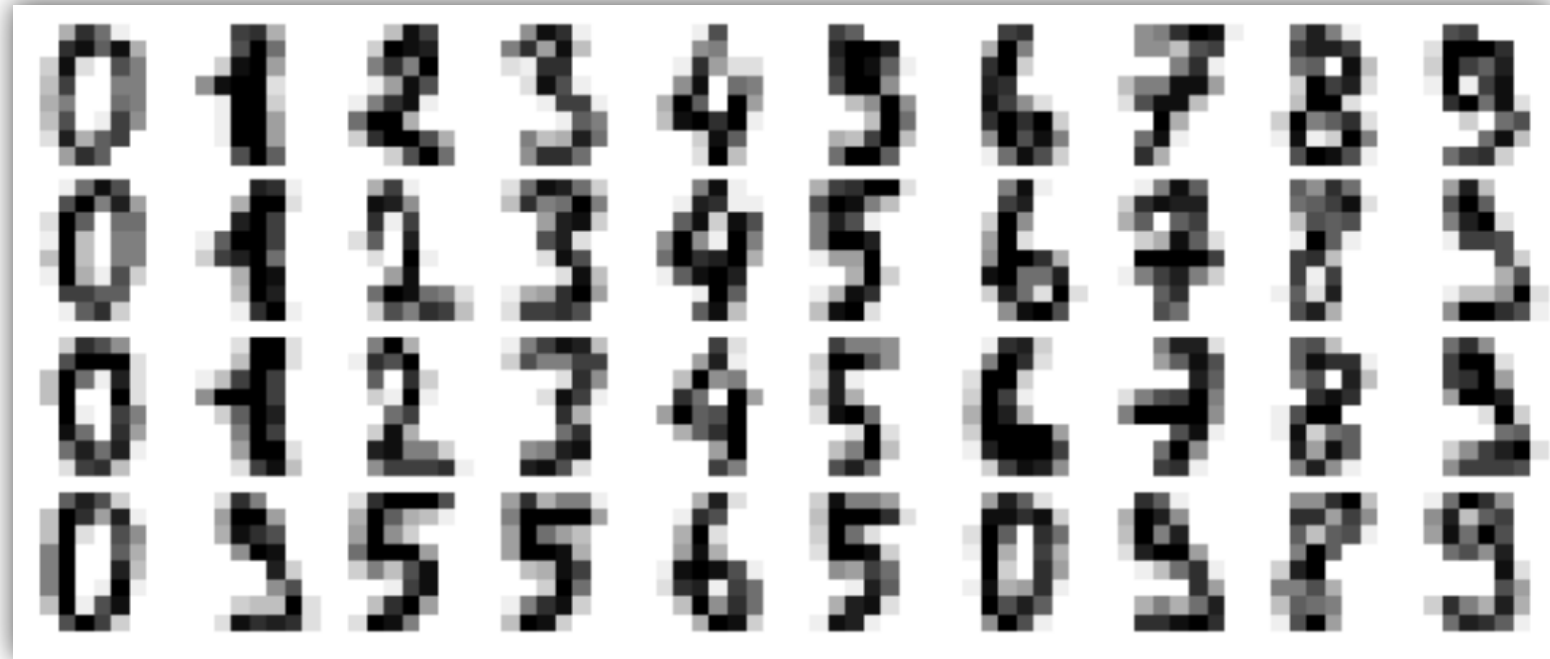
Optimal:  
20 components explain  
90% of variance



# PCA as Noise Filtering

Any components with variance larger than the effect of the noise should be relatively unaffected by the noise

If data is reconstructed using only the largest subset of principal components, it will keep the signal and throw out the noise.



# PCA as Noise Filtering

```
np.random.seed(42)  
noisy = np.random.normal(digits.data, 4)  
plot_digits(noisy)
```

Adding random noise to  
the data



# PCA as Noise Filtering

```
pca = PCA(0.50).fit(noisy)  
pca.n_components_
```



```
components = pca.transform(noisy)  
filtered = pca.inverse_transform(components)  
plot_digits(filtered)
```

# PCA

Start with PCA to visualize the relationship between points to understand

- the main variance
- the intrinsic dimensionality by plotting explained variance ratio

Cons

- affected by outliers in the data





# Case Study: Eigenfaces

Dataset: Sklearn built-in LFW dataset

Notebook: ML07\_PCA\_Example.ipynb

Steps:

Download the dataset

Explore the dataset

Prepare the dataset

Build the model

Predict on testing data

Report insights



# Gaussian Mixture Models (GMM)

# Gaussian Mixture Models (GMM)

Unsupervised algorithm

Searches for a pre-determined number of clusters within an unlabeled multidimensional dataset

Two main assumptions

- Cluster center is arithmetic mean of all points belonging to the cluster
- Each point is closer to its own cluster center than other cluster centers



# Case Study: Handwritten Digits

Dataset: Digits dataset

Notebook: ML09\_GMM\_Example.ipynb

To do:

Download the dataset

Explore the dataset

Prepare the dataset

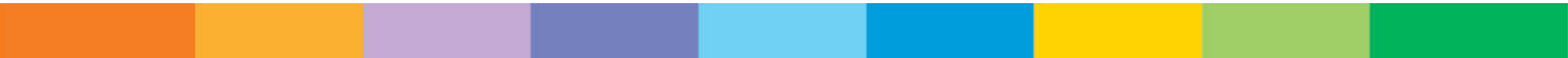
Build the model

Predict on testing data

Report insights

# Benchmarking Algorithms

Task 10: Benchmark all the algorithms we've seen so far for a specific classification problem



# Further Reading

Scikit-learn documentation

<https://scikit-learn.org/>

Python Data Science Handbook by *Jake Vanderplas*