CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

scikit learn

Machine Learning

Machine Learning

Lecture: Learning Curves

Ted Scully

Bias Vs Variance

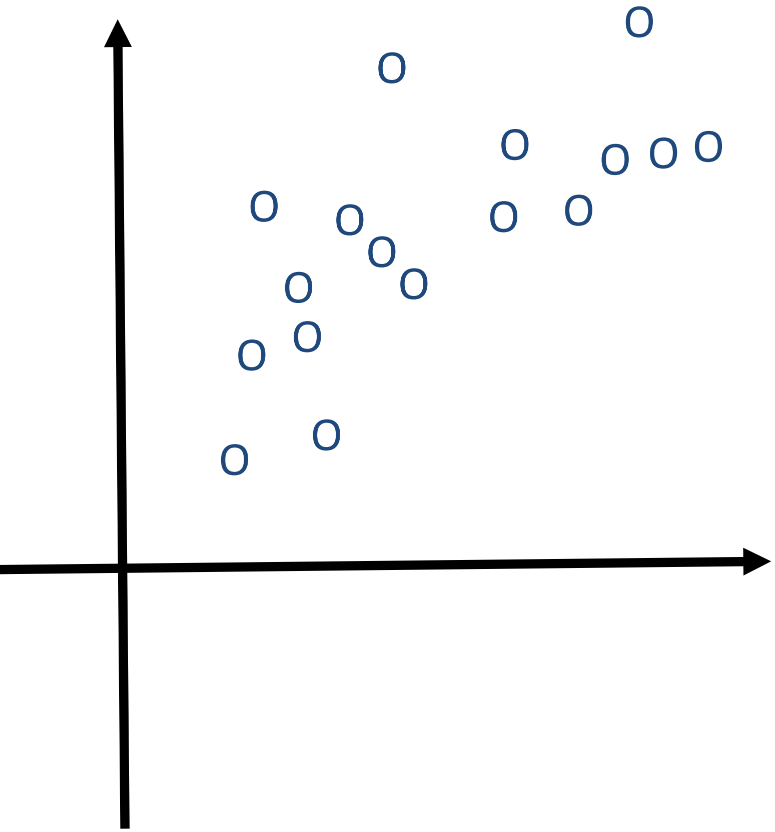
- Normally, you want your model to fit the **training data** as best as possible and also generalise as best as possible to **unseen data**.

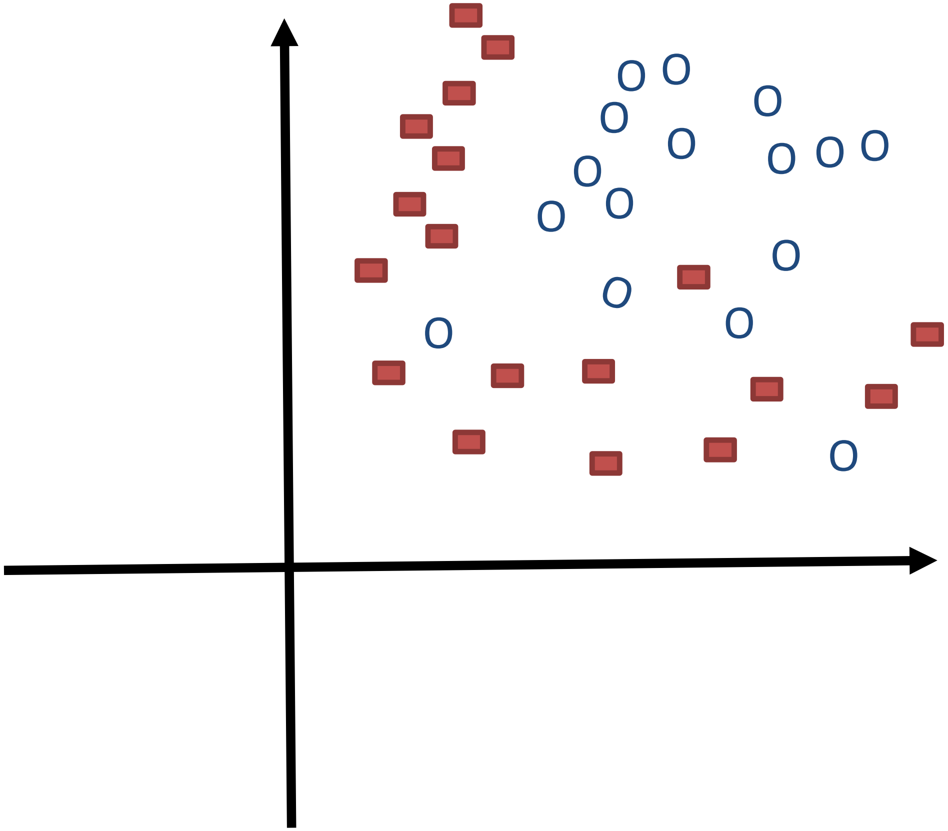
- In many situations it is difficult to achieve both simultaneously.

- *Bias* is the inability of an ML model to capture the **patterns in the underlying training data**.
  - Algorithms with **high bias** typically underfit their training data, failing to capture important features of the data.

- *Variance* is the sensitivity of the ML model to changes in the dataset.
  - **High-variance** models may be able to represent their training set well, but may exhibit high variance in the results when tested on different samples of data.

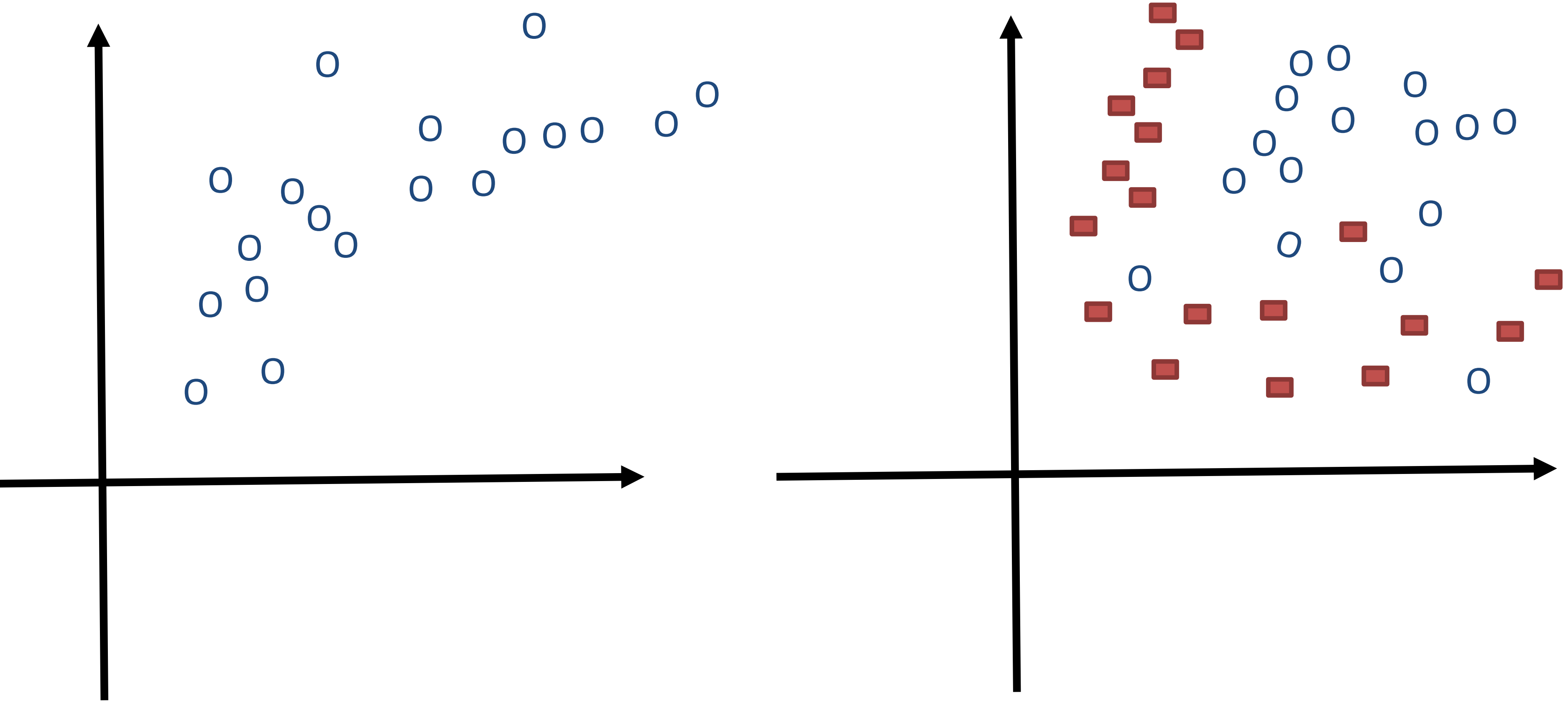
High Bias





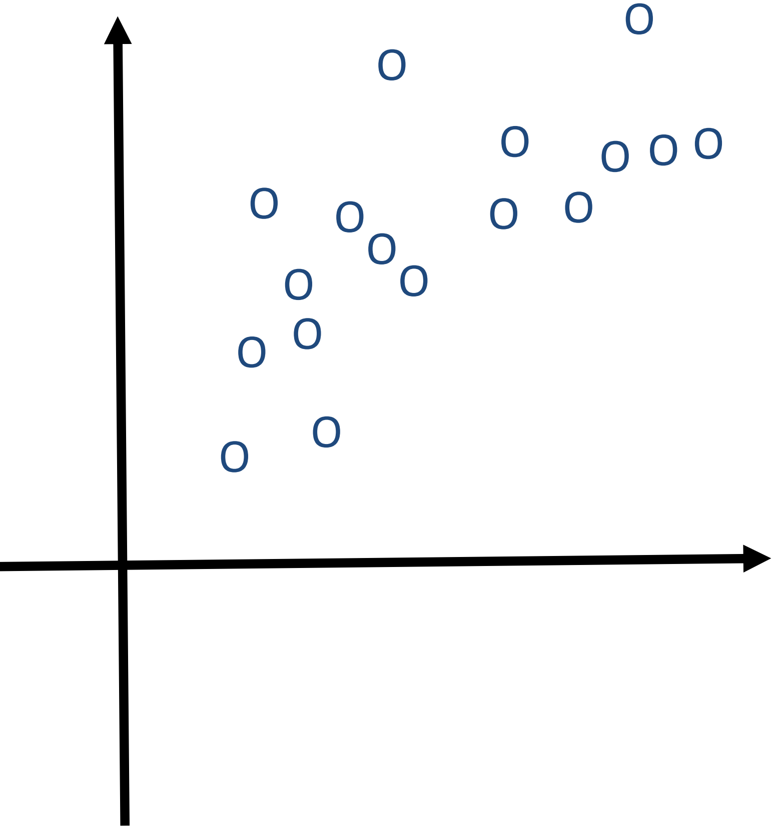
- Algorithms with **high bias** may typically produce simpler models that tend to underfit their training data, failing to capture important features of the data.

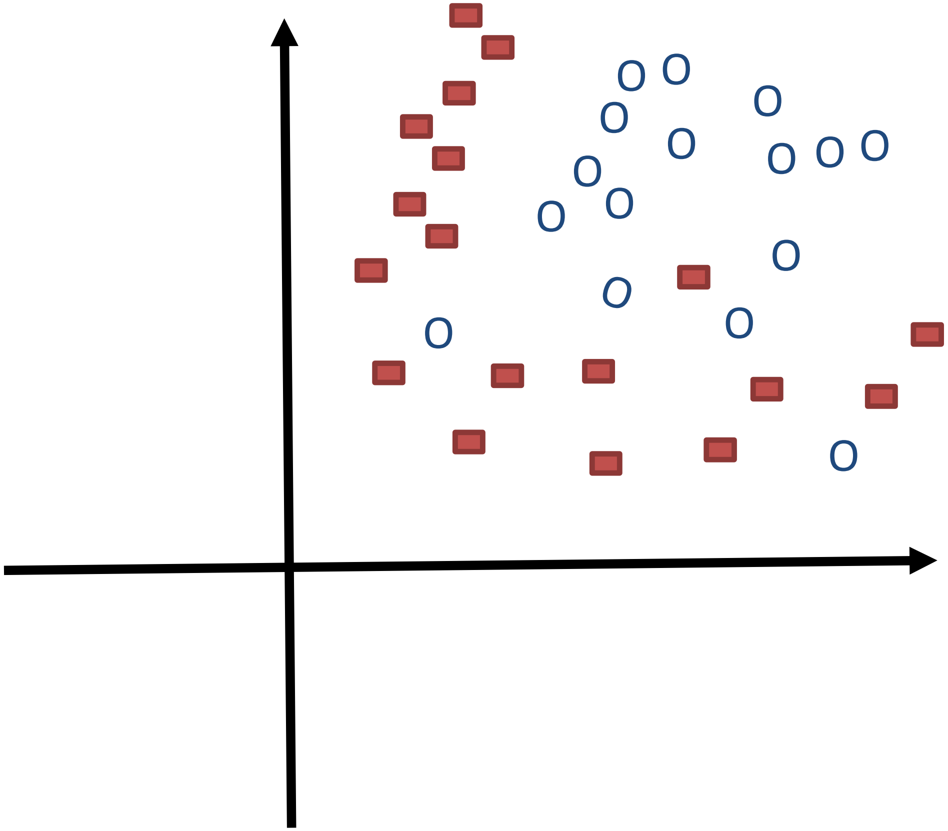
High Variance



- **High-variance** models may be able to represent their training set well, but are at risk of overfitting to noisy or unrepresentative training data.

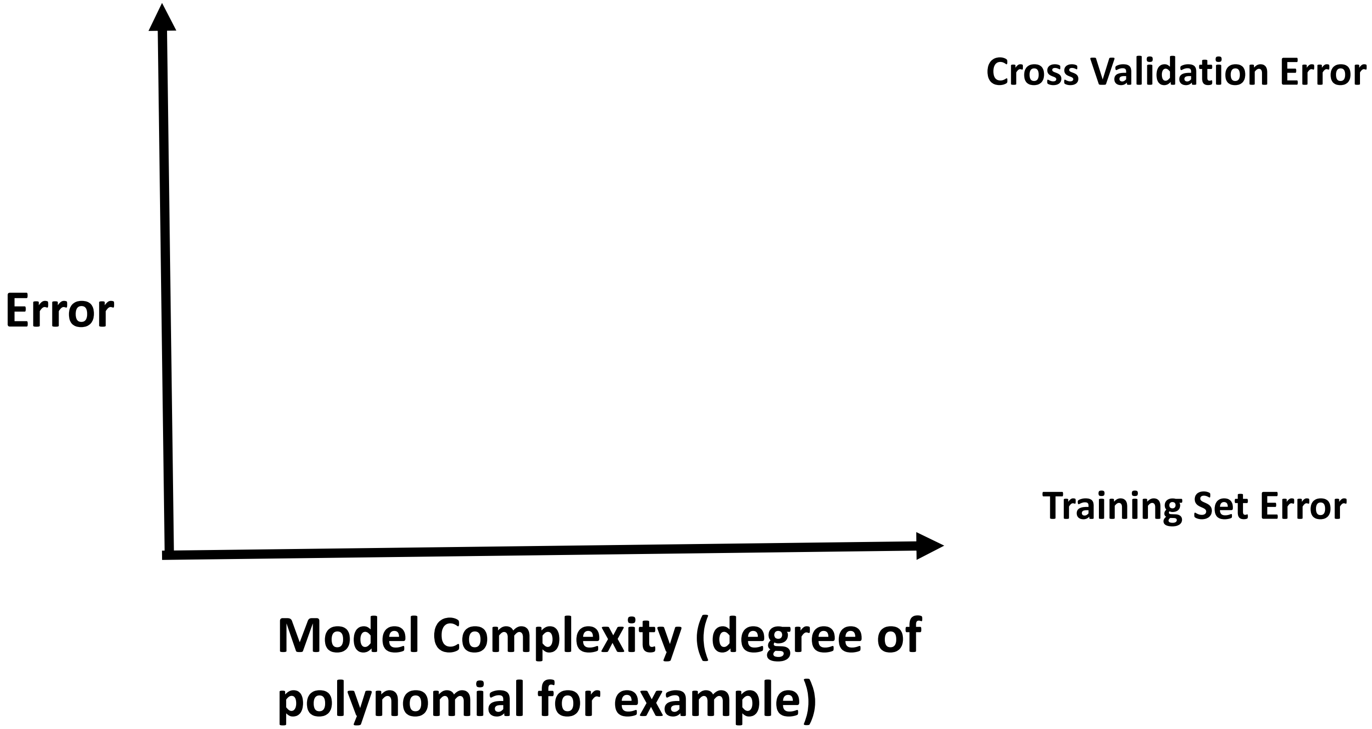
Balance between Bias and Variance

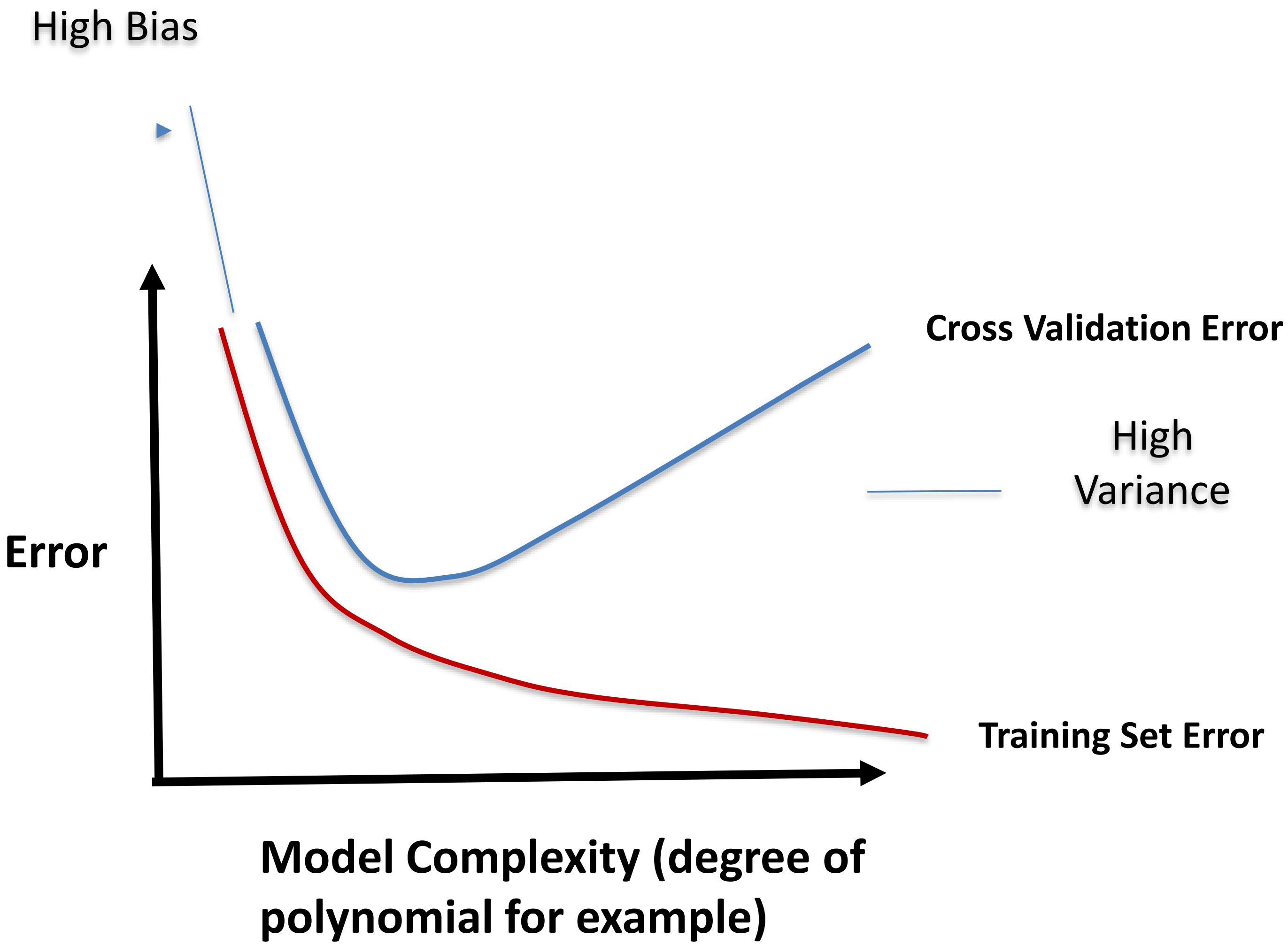




Bias and Variance

- Next let's consider two metrics:
  - Error on **training set** (error after applying model to training set)
  - Error on **cross validation** (error rate after performing CV using model)
  - On the model what area corresponds to high bias and high variance?

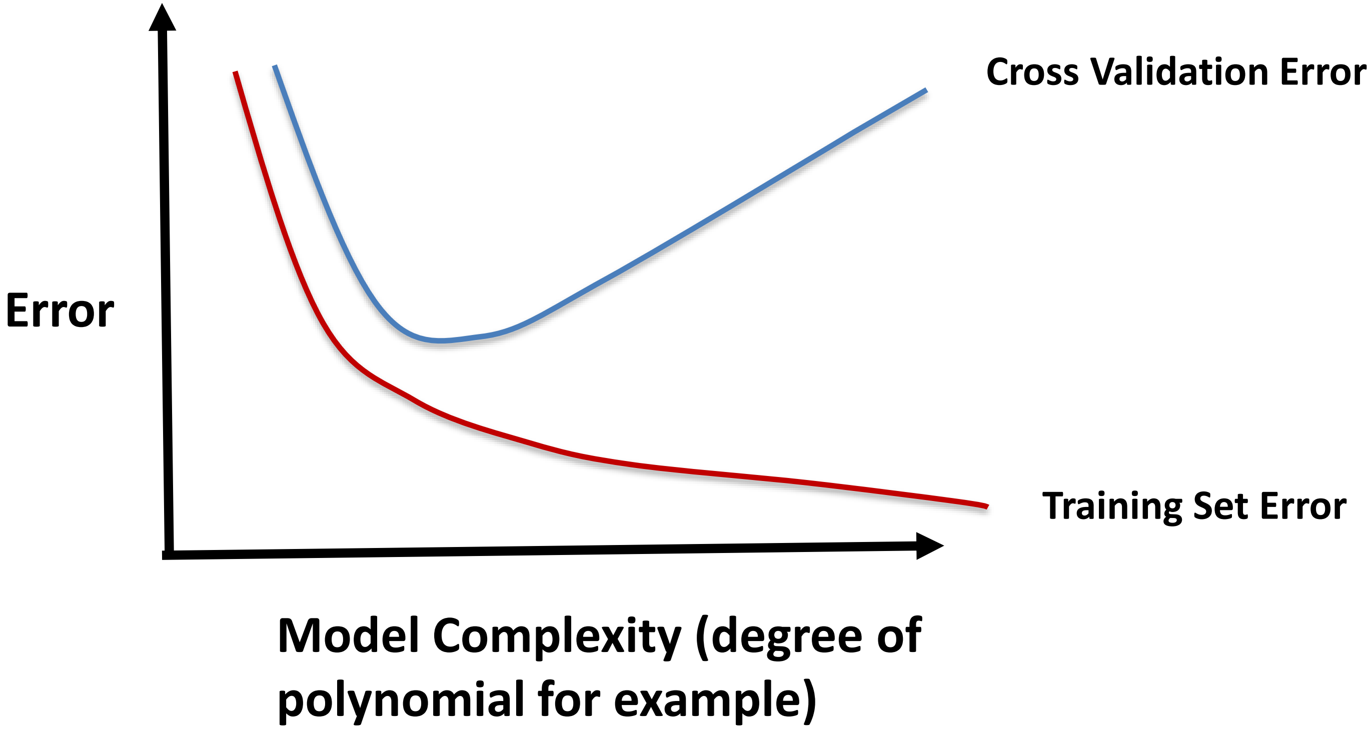


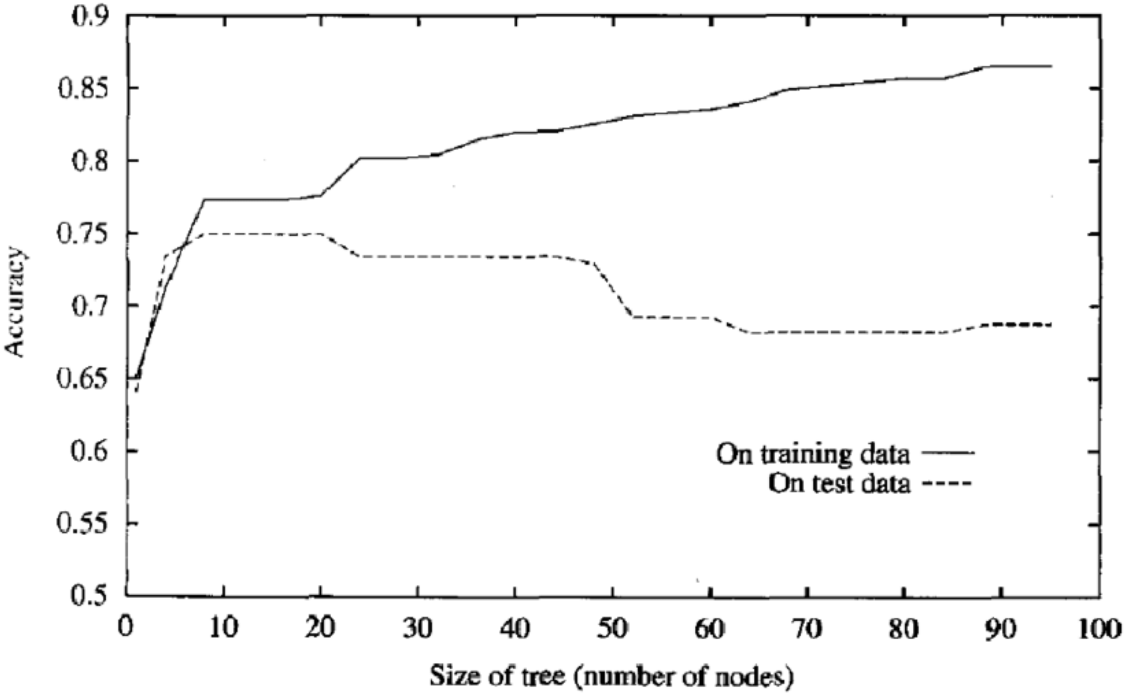

High Bias
Cross Validation Error
High
Variance
Error
Training Set Error
Model Complexity (degree of polynomial for example)

Bias and Variance

1. High Bias – Training Error and Cross Validation Error is also typically high.

2. High Variance – Cross Validation Error High – Training Error Low



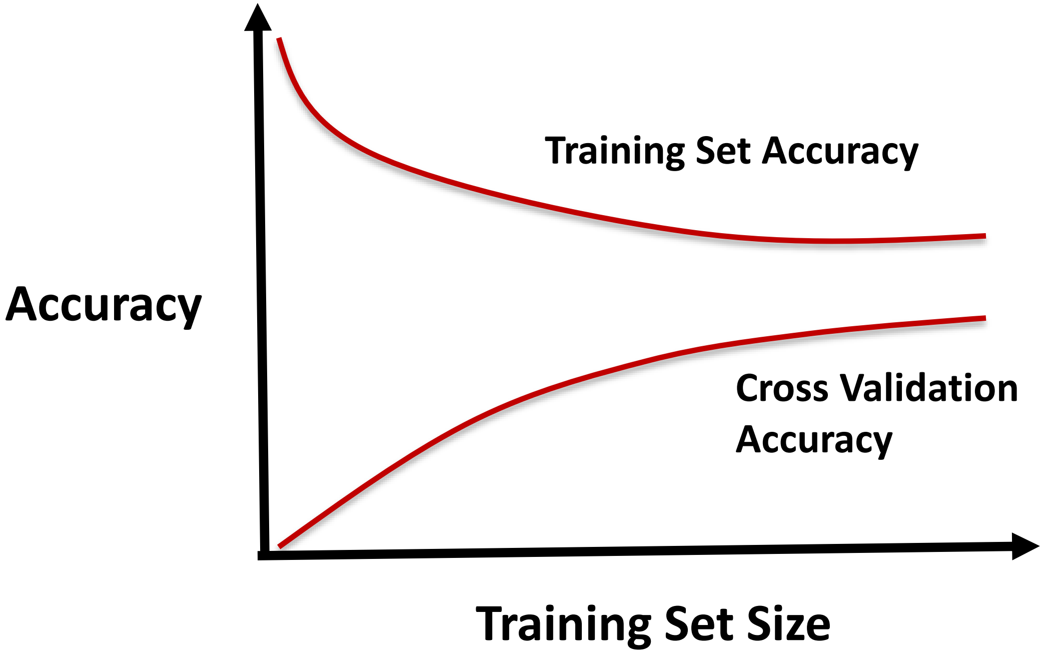

0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
Accuracy
0
10
20
30
40
50
60
70
80
90
100
Size of tree (number of nodes)
On training data
On test data

Learning Curves

- A learning curve **plots a models performance on the training and unseen data** as a function of the training set size.
- It is a useful tool for detecting whether the model suffers from **high variance** or **high bias**, which in turn can help us identify what action may help improve the overall accuracy of the algorithm.

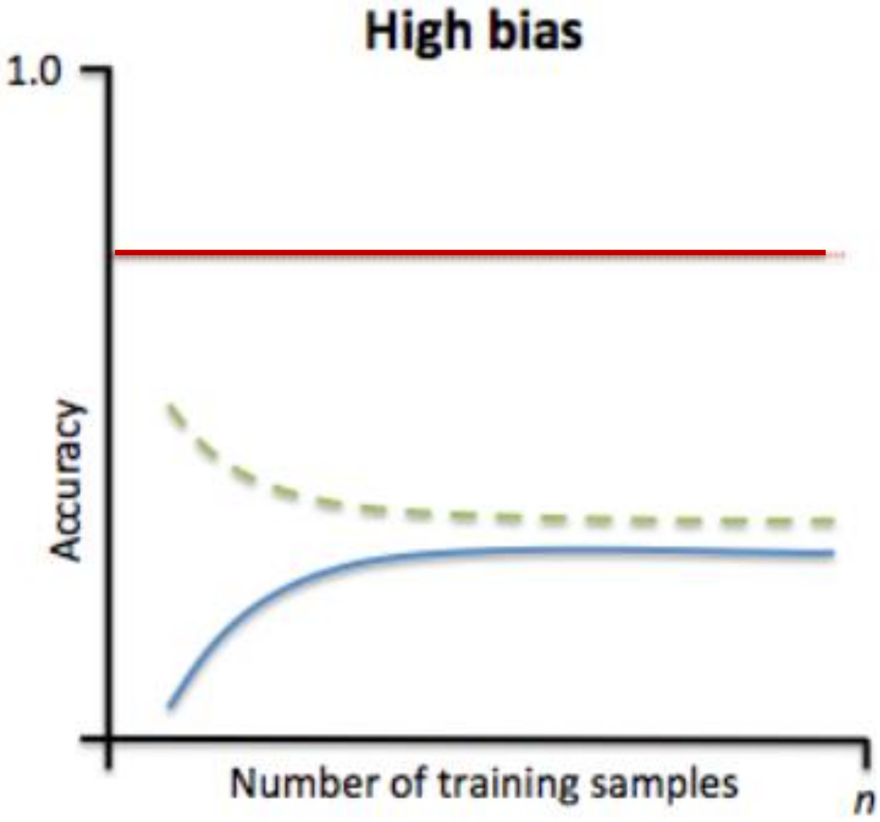
Learning Curves

- Can help diagnose if an algorithm is suffering from high bias or high variance.
- Can also give insight into when the model converges and if more data would be helpful in improving model performance.
- Notice we specify the accuracy on the Y axis.



Learning Curves – High Bias

- With high bias the performance of the model on the training and cross validation set will be similar (**both will exhibit a poor accuracy or a high error**).
- Another interesting observation is that the addition of **additional data in a high bias scenario typically won't help improve the model performance**.



So knowing that your algorithm suffers from high bias is useful because you won't waste time trying to gather additional data in order to improve performance.

Dealing with High Bias

If encountering high bias, the following steps may be helpful:

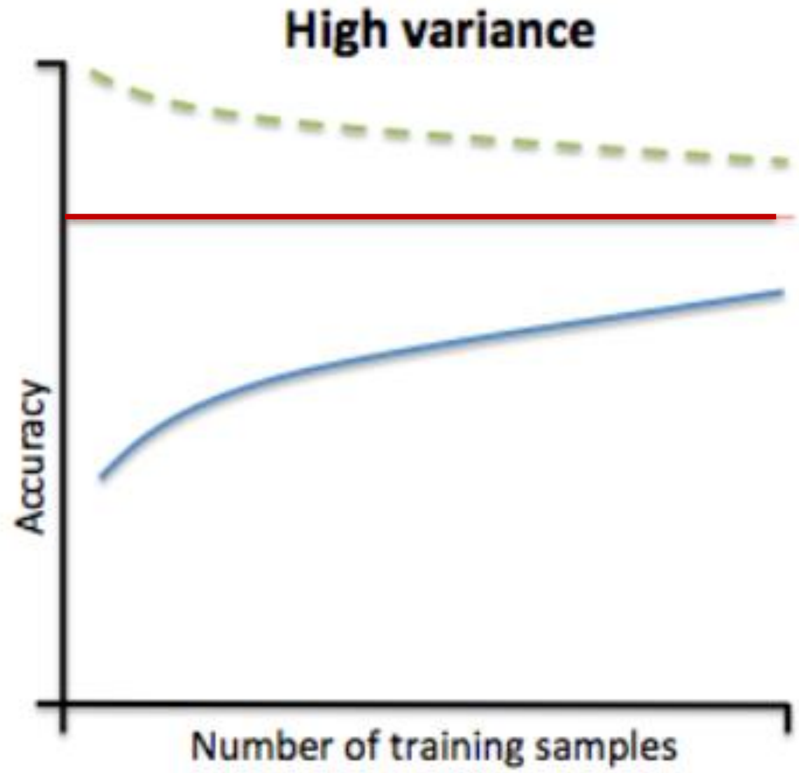
1.Use a more sophisticated model. Adding complexity to the model can help improve on bias. For a polynomial fit, this can be accomplished by increasing the degree *d*.

2.Add more features. Adding additional features to the data could help improve a high-bias estimator. The existing features may not be strong enough predictors of the final class.

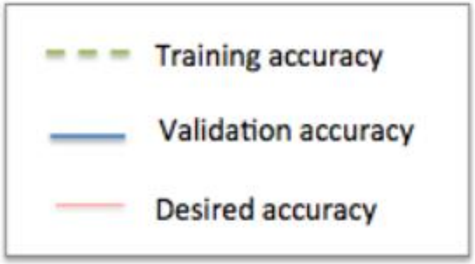
3.Use fewer samples. This will <u>not</u> improve the performance and is just an observation. A high-bias algorithm can attain nearly the same error with a smaller number of training samples. For algorithms which are computationally expensive, reducing the training sample size can lead to very large improvements in speed.

Learning Curves – High Variance

- If you end up with a relatively large gap between CV and training accuracy it is a strong indication that you are suffering form high variance (where the training accuracy is high but unseen accuracy significantly lower).
- In a high variance case it may often be the case that accuracy values have not converged and in many cases addition of more training data can help improve the performance of the model.



Additional data will often help a high variance problem. Reducing the number of features can also help improve performance of an algorithm suffering form high variance.



Dealing High Variance

If encountering high variance, the following steps may help :

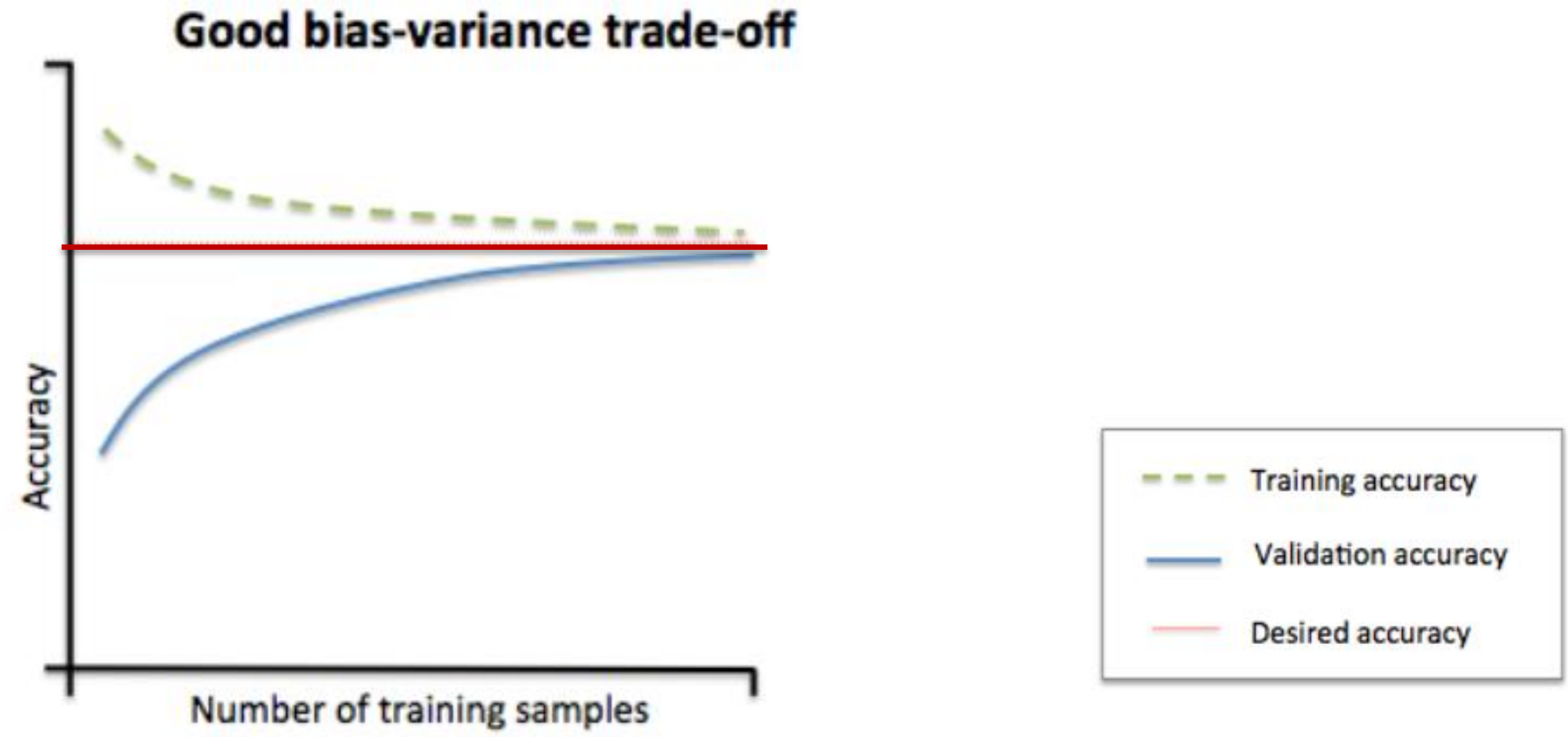
1. Try to reduce the complexity of your model. Increasing the regularization parameter, pruning a decision tree, increase value of k in kNN, reducing c value in SVM, etc. These are all examples that can reduce variance in a model.

2. Use fewer features. Using a feature selection technique may be useful, and decrease the over-fitting of the estimator.

3. Use more training samples. Adding training samples can reduce the effect of over-fitting, and lead to improvements in a high variance estimator.

Learning Curves

- The graph shown below shows a good bias-variance trade-off.
- Notice that the accuracy on the training and validation data tend to converge and also plateaux.



Learning Curves in Scikit - Learn

- To build a learning curve in scikit learn we can use the **learning_curve** module in sklearn.learning_curve.

- **estimator** : ML algorithm that implements the “fit” and “predict” methods an object of that type which is cloned for each validation.

- **X** : training data (n_samples, n_features)

- **y**: class label vector length n_samples

Learning Curves in Scikit - Learn

- Parameters of learning_curve :

- **cv** : Determines the cross-validation splitting strategy. Possible inputs for cv are:
  - None, to use the default 3-fold cross-validation,
  - integer, to specify the number of folds.
  - An object to be used as a cross-validation generator.

- **train_sizes** : specifies how we should divide the training set for each point in the learning curve.
  - We can specify this using **np.linspace** (np.linspace(2.0, 3.0, num=5) will return [2. , 2.25, 2.5 , 2.75, 3.]). If we have 1000 training examples and wish our learning curve to have 5 values between 100 and 1000 then we can specify **np.linspace(0.1, 1, 5)** -> **[0.1 0.325 0.55 0.775 1.]**

Learning Curves in Scikit - Learn

- Parameters of learning_curve :
- **train_sizes** : specifies how we should divide the training set for each point in the learning curve.
  - We can specify this using **np.linspace** (np.linspace(2.0, 3.0, num=5) will return [2. , 2.25, 2.5 , 2.75, 3.]). If we have 1000 training examples and wish our learning curve to have 10 values between 100 and 1000 then we can specify
  - **np.linspace(0.1, 1, 10)**
  - **[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.]**

Learning Curves in Scikit - Learn

- The learning_curve method then returns the following:
  - **train_sizes_abs**: Numbers of training examples that has been used to generate the learning curve.
  - **train_scores** : array, shape (n_ticks, n_cv_folds) – Scores on training set
  - **test_scores** : array, shape (n_ticks, n_cv_folds) – CV score on test set

- It is import to understand that both train_scores and test_scores are both 2D NumPy arrays.

Learning Curves in Scikit - Learn

- The learning_curve method then returns the following:
  - **test_scores** : array, shape (n_ticks, n_cv_folds) – CV score on test set

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB

from sklearn.datasets import load_iris
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

iris = load_iris()
X, y = iris.data, iris.target

# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
custom_cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)


estimator = GaussianNB()

title = "Learning Curve Iris (NB)"
plt.title(title)
ylim=(0.0, 1.01)
```

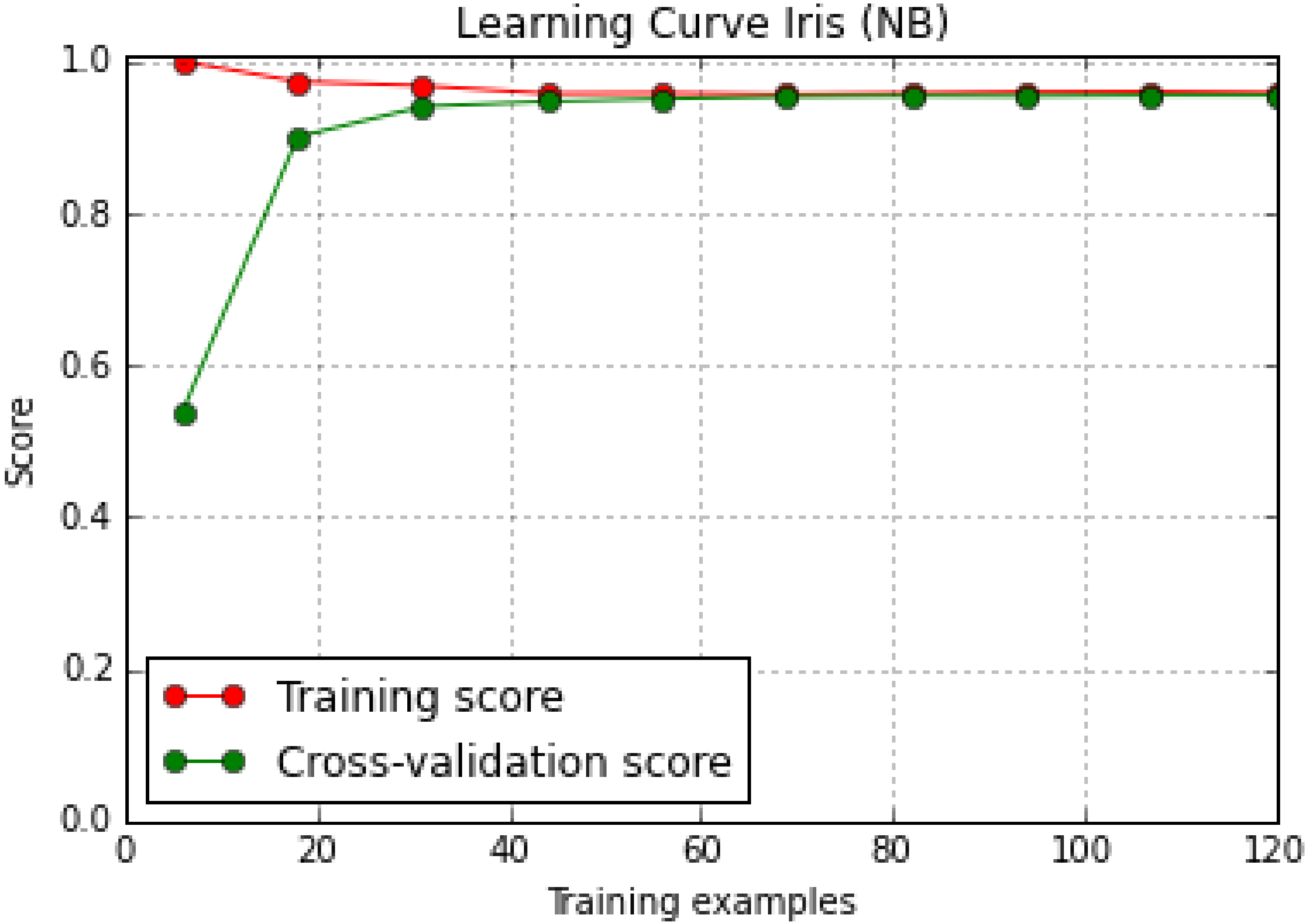
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB

from sklearn.datasets import load_iris
from sklearn.model_selection import learning
from sklearn.model_selection import ShuffleS

iris = load_iris()
X, y = iris.data, iris.target

# Cross validation with 100 iterations to get sm
# score curves, each time with 20% data rando
custom_cv = ShuffleSplit(n_splits=100, test_s


estimator = GaussianNB()

title = "Learning Curve Iris (NB)"
plt.title(title)
ylim=(0.0, 1.01)
```

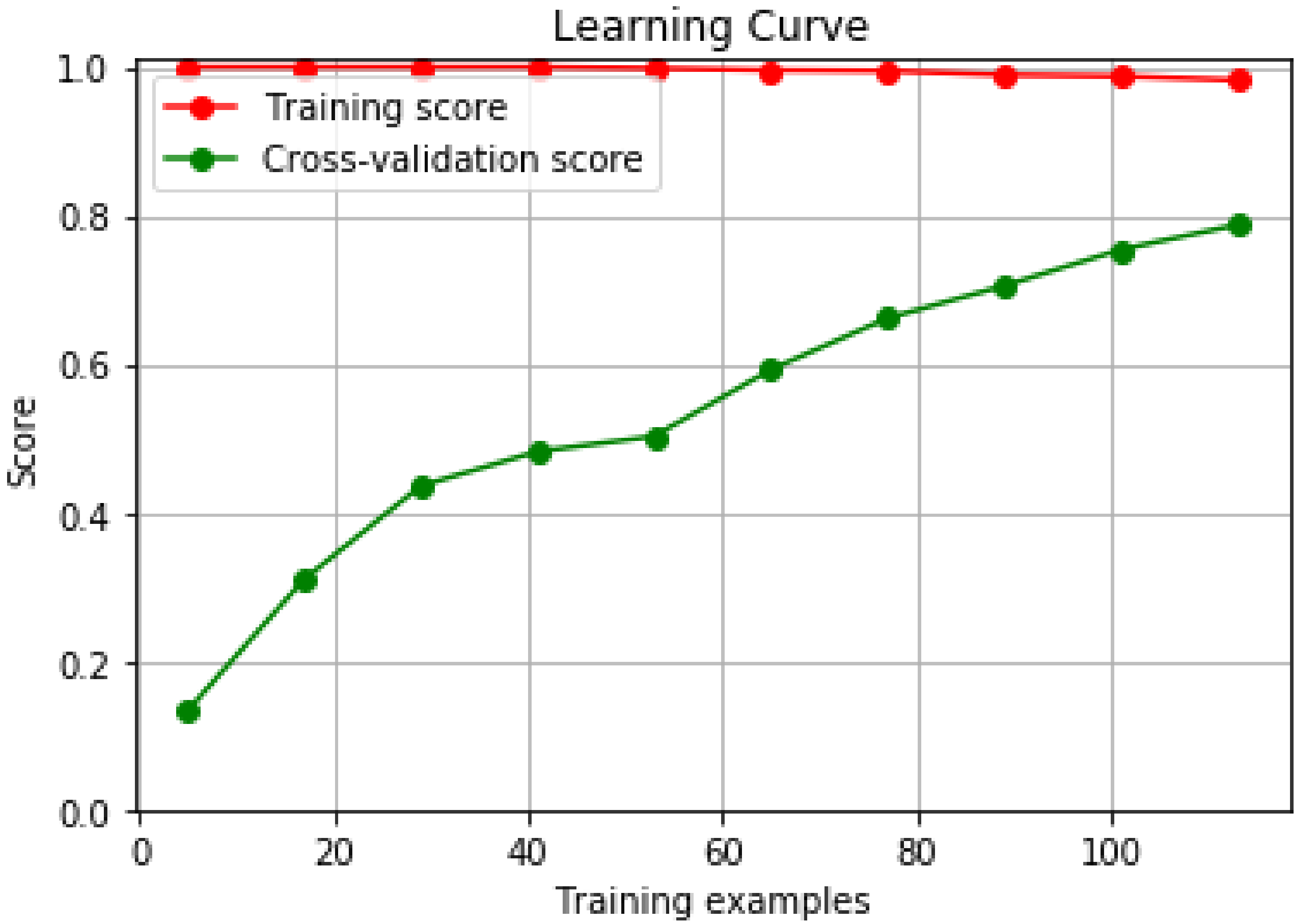
Notice we use a large number of folds (100) to make sure that the results are smooth. If we were to use normal cross fold validation with 100 folds, we may risk not having enough data to split into 100 different folds.

ShuffleSplit takes a slightly different approach. For each fold it randomly select 20% of the data for test and the rest for training. So it doesn't suffer from the same problem.

```
plt.ylim(ylim)
plt.xlabel("Training examples")
plt.ylabel("Score")

train_sizes=np.linspace(.05, 1.0, 10)

train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
train_sizes=train_sizes, cv = custom_cv)

train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
plt.grid()


plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

plt.legend(loc="best")
plt.show()
plt.ylim(ylim)
plt.xlabel("Training examples
plt.ylabel("Score")

train_sizes=np.linspace(.05,

train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
train_sizes=train_sizes, cv = custom_cv)

train_scores_mean = np.mean(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
plt.grid()


plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

plt.legend(loc="best")
plt.show()
```

Remember axis = 1 operates on the horizontal axis and calculates the mean for each row.
Each row corresponds to one tick (first row might correspond to 10% of data, the next row 20% of data, etc) and each value in a row is the accuracy achieved for each iteration of the cross fold (so in this example, each row would have 100 values).


Learning Curve Iris (NB)
Score
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
Training examples
Training score
Cross-validation score


Learning Curve
Training score
Cross-validation score
Score
Training examples
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100