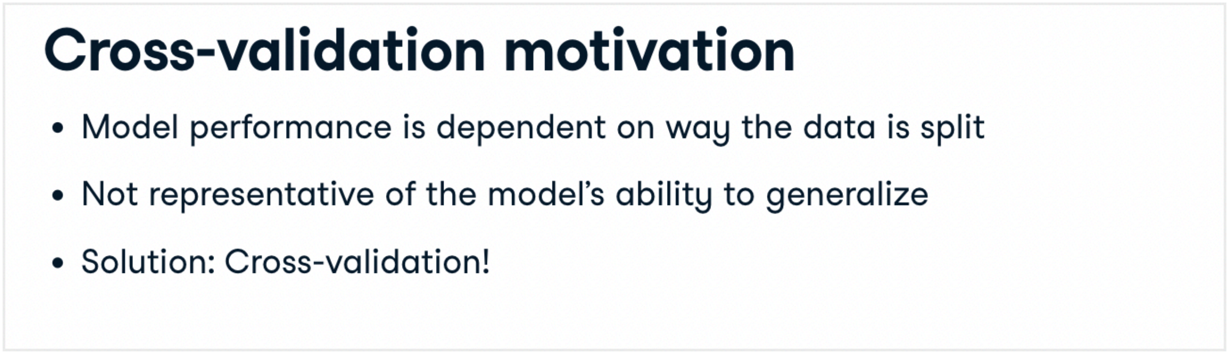
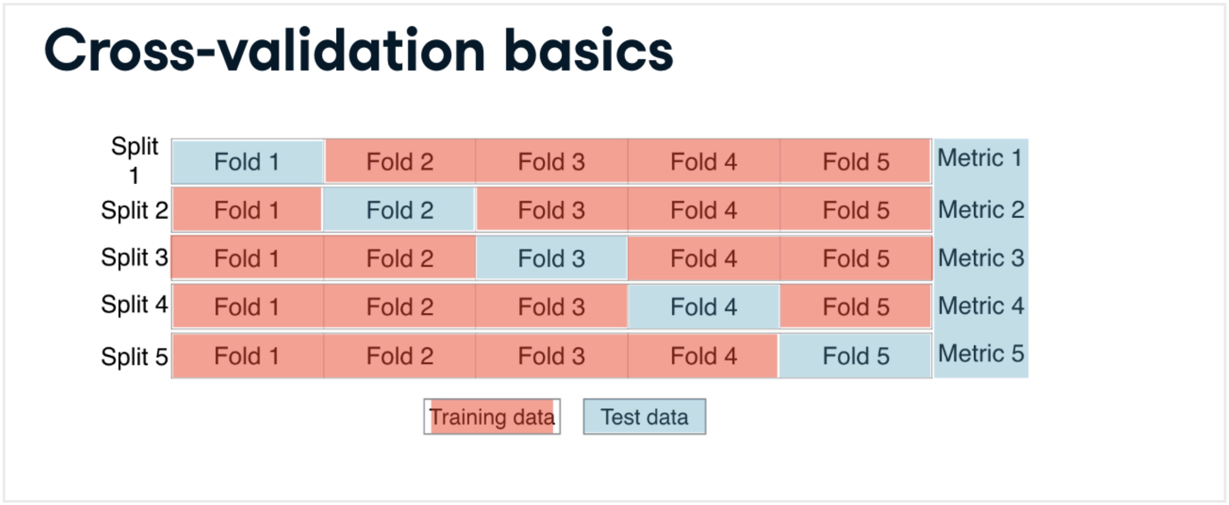
Supervised Learning - 2

Cross Validation
Cross-validation is a vital step in evaluating a model. It maximizes the amount of data that is used to train the model, as during the course of training, the model is not only trained, but also tested on all of the available data.



Cross-validation motivation

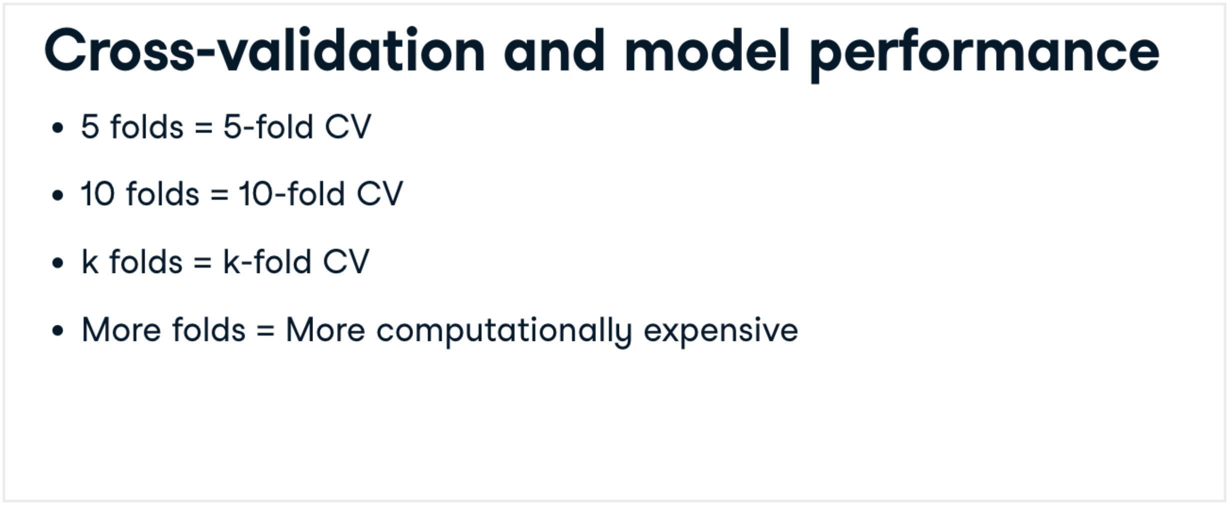
- Model performance is dependent on way the data is split
- Not representative of the model's ability to generalize
- Solution: Cross-validation!



Cross-validation basics

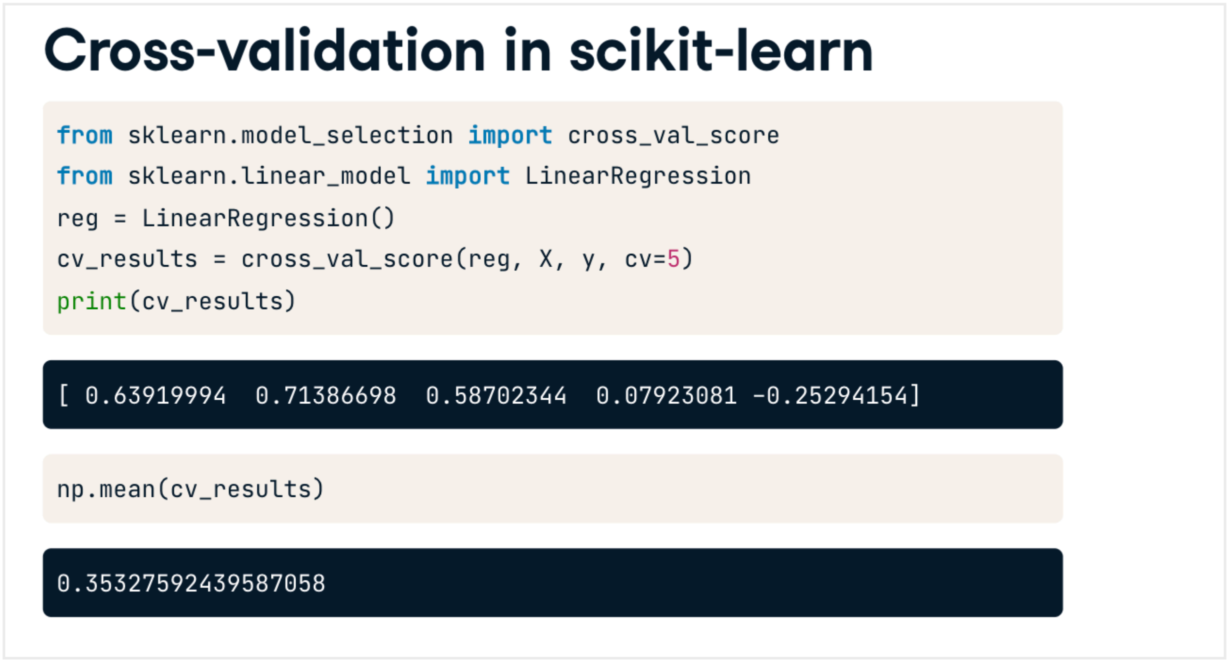
Split 1	🟦 Fold 1	🟥 Fold 2	🟥 Fold 3	🟥 Fold 4	🟥 Fold 5	Metric 1
Split 2	🟥 Fold 1	🟦 Fold 2	🟥 Fold 3	🟥 Fold 4	🟥 Fold 5	Metric 2
Split 3	🟥 Fold 1	🟥 Fold 2	🟦 Fold 3	🟥 Fold 4	🟥 Fold 5	Metric 3
Split 4	🟥 Fold 1	🟥 Fold 2	🟥 Fold 3	🟦 Fold 4	🟥 Fold 5	Metric 4
Split 5	🟥 Fold 1	🟥 Fold 2	🟥 Fold 3	🟥 Fold 4	🟦 Fold 5	Metric 5

🟥 Training data 🟦 Test data



Cross-validation and model performance

- 5 folds = 5-fold CV
- 10 folds = 10-fold CV
- k folds = k-fold CV
- More folds = More computationally expensive



Cross-validation in scikit-learn

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=5)
print(cv_results)
```

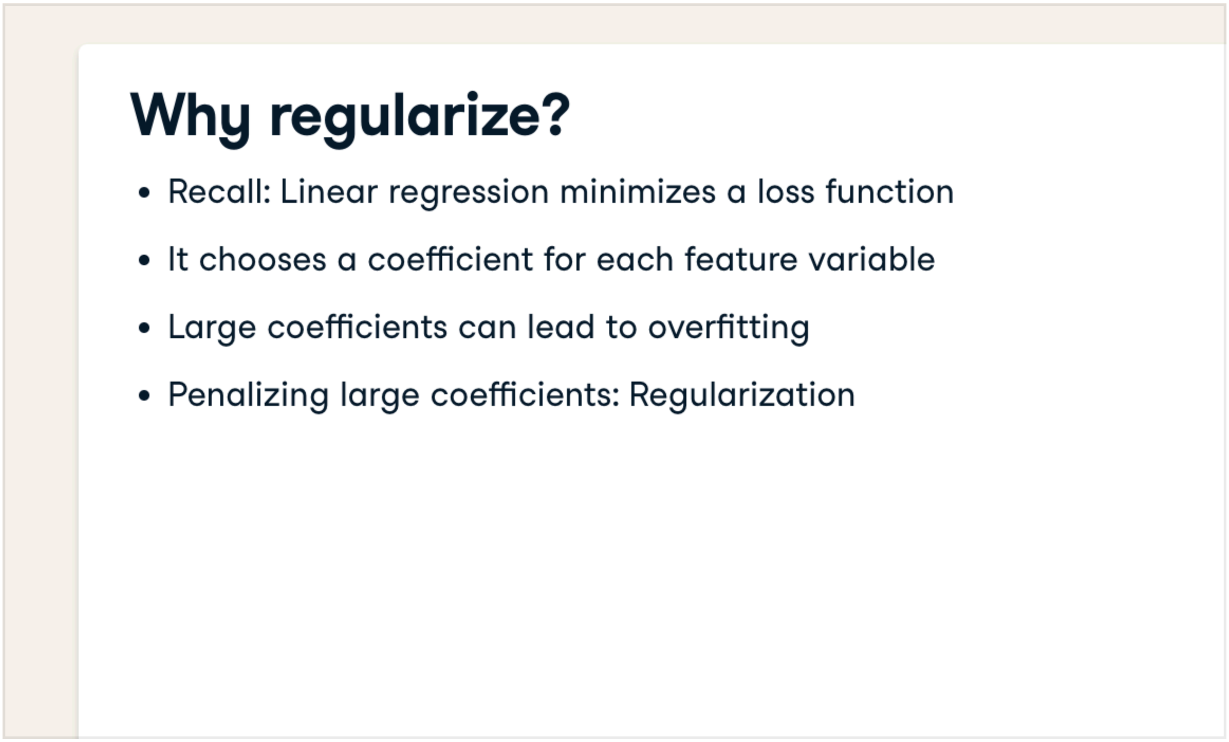
```
[ 0.63919994  0.71386698  0.58702344  0.07923081 -0.25294154]
```

```
np.mean(cv_results)
```

```
0.35327592439587058
```

cv - No of folds
error metric - R^2.
We pass in the model and the training data along with the number of cross folds to the cross validation score function.

Regularized regression



Why regularize?

- Recall: Linear regression minimizes a loss function
- It chooses a coefficient for each feature variable
- Large coefficients can lead to overfitting
- Penalizing large coefficients: Regularization

Ridge regression

- Loss function = OLS loss function +

$$\alpha * \sum_{i=1}^{n} {a_i}^2$$

- Alpha: Parameter we need to choose
- Picking alpha here is similar to picking k in k-NN
- Hyperparameter tuning (More in Chapter 3)
- Alpha controls model complexity
    - Alpha = 0: We get back OLS (Can lead to overfitting)
    - Very high alpha: Can lead to underfitting

Ridge regression in scikit-learn

```python
from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, random_state=42)
ridge = Ridge(alpha=0.1, normalize=True)
ridge.fit(X_train, y_train)
ridge_pred = ridge.predict(X_test)
ridge.score(X_test, y_test)
```

```
0.699693827512731 79
```

Lasso regression

- Loss function = OLS loss function +

$$\alpha * \sum_{i=1}^{n} |a_i|$$

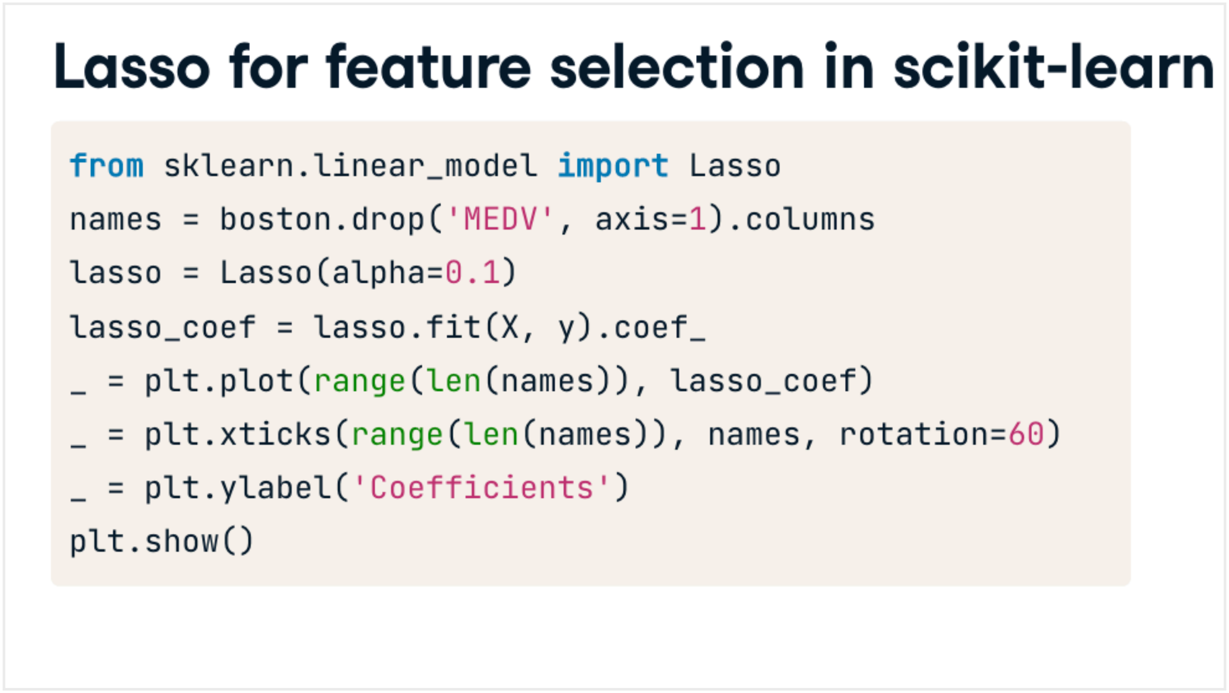
Lasso regression in scikit-learn

```
from sklearn.linear_model import Lasso
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.3, random_state=42)
lasso = Lasso(alpha=0.1, normalize=True)
lasso.fit(X_train, y_train)
lasso_pred = lasso.predict(X_test)
lasso.score(X_test, y_test)
```

```
0.5950229535328551
```

Lasso regression for feature selection

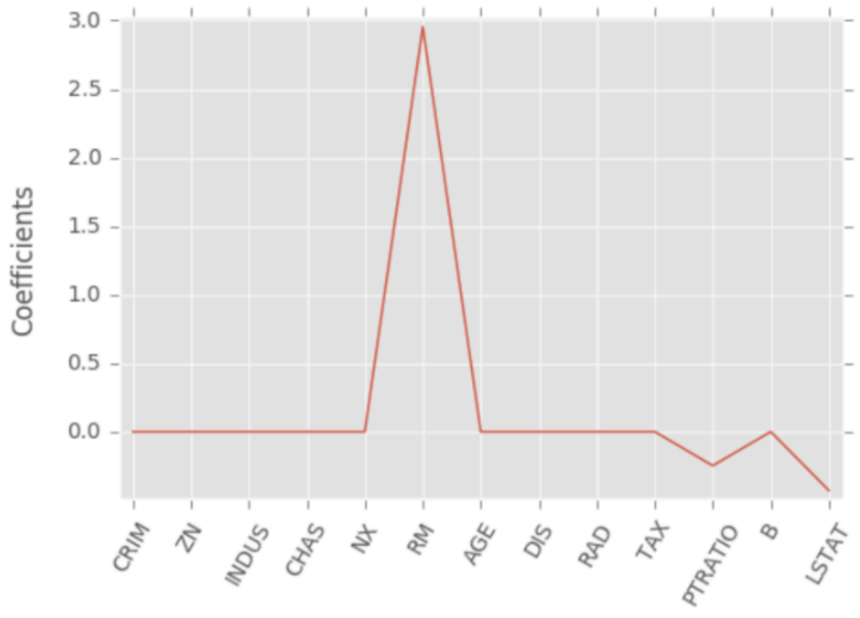
- Can be used to select important features of a dataset
- Shrinks the coefficients of less important features to exactly 0



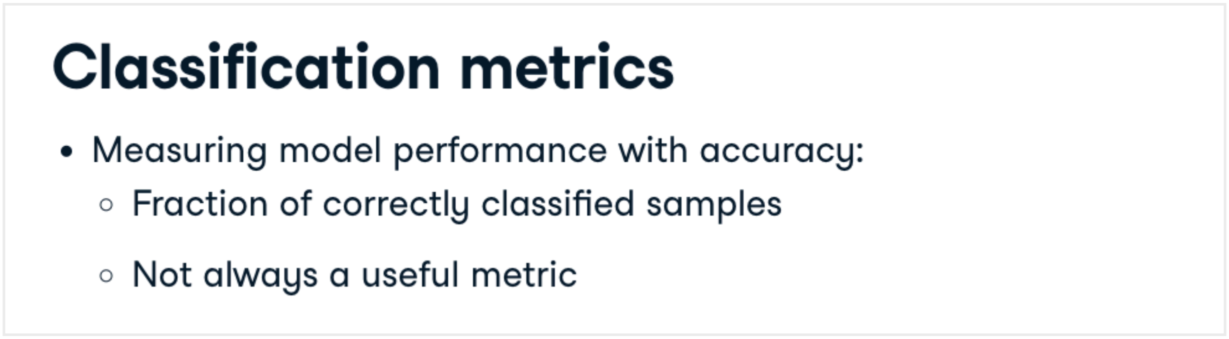
Lasso for feature selection in scikit-learn

```python
from sklearn.linear_model import Lasso
names = boston.drop('MEDV', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
_ = plt.plot(range(len(names)), lasso_coef)
_ = plt.xticks(range(len(names)), names, rotation=60)
_ = plt.ylabel('Coefficients')
plt.show()
```

Lasso for feature selection in scikit-learn

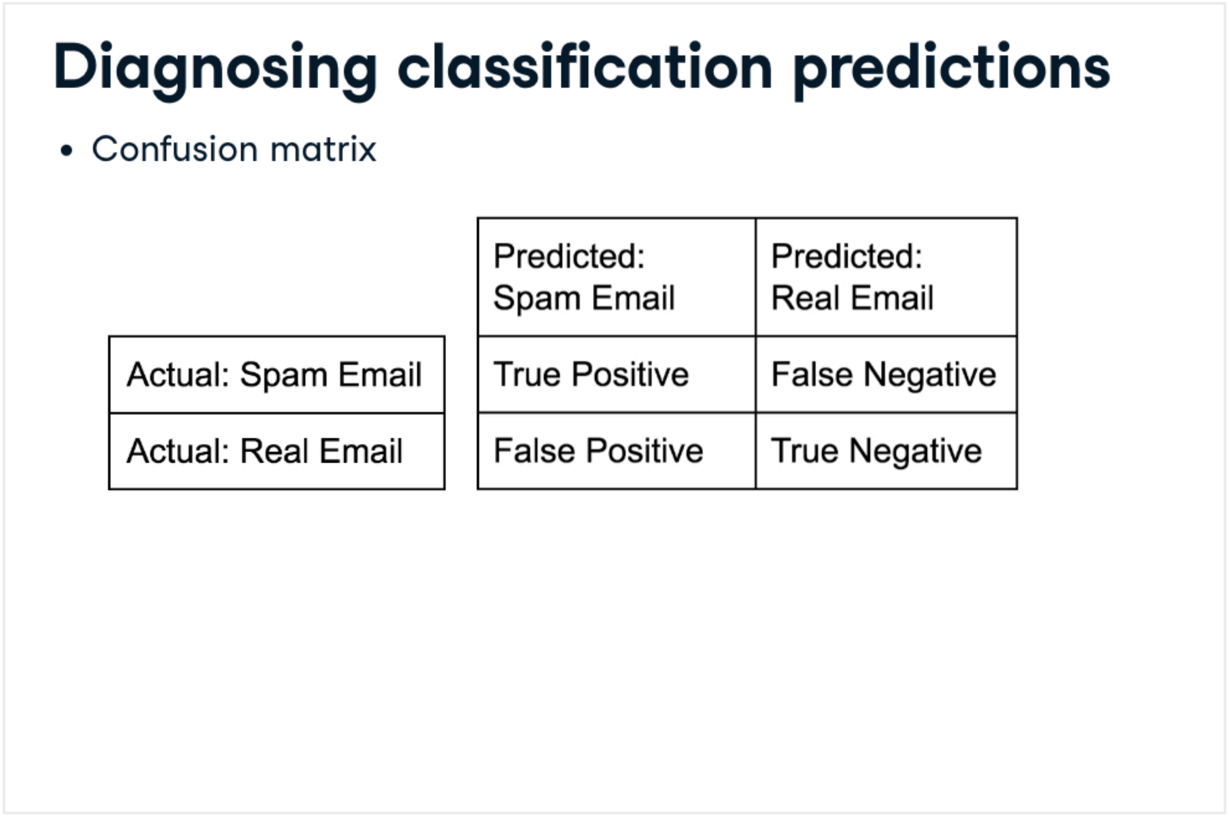


Classification Metrics



Classification metrics

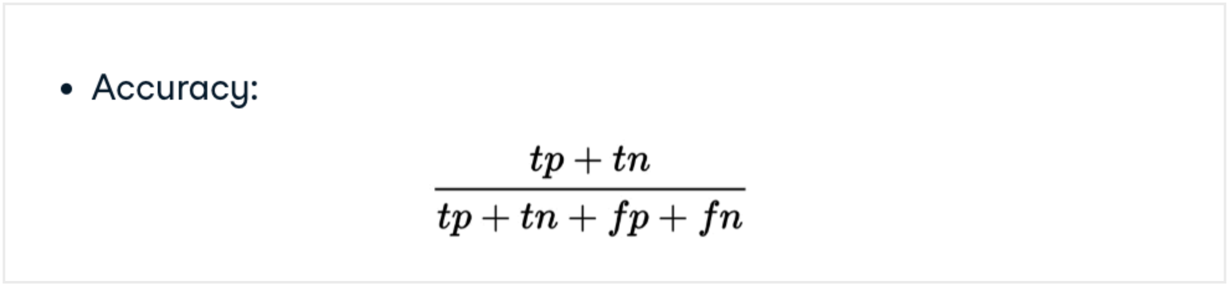
- Measuring model performance with accuracy:
  - Fraction of correctly classified samples
  - Not always a useful metric



Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	True Positive	False Negative
Actual: Real Email	False Positive	True Negative



- Accuracy:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Precision: tp/(tp + fn) Out of all the positive classified labels, how many are actually positive.
Recall: tp/(tp+fn): Out of all the actual positive labels, how many were currently classified.

- Precision $\frac{tp}{tp+fp}$
- Recall $\frac{tp}{tp+fn}$
- F1score: $2 \cdot \frac{precision*recall}{precision+recall}$
- High precision: Not many real emails predicted as spam
- High recall: Predicted most spam emails correctly

```python
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
knn = KNeighborsClassifier(n_neighbors=8)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

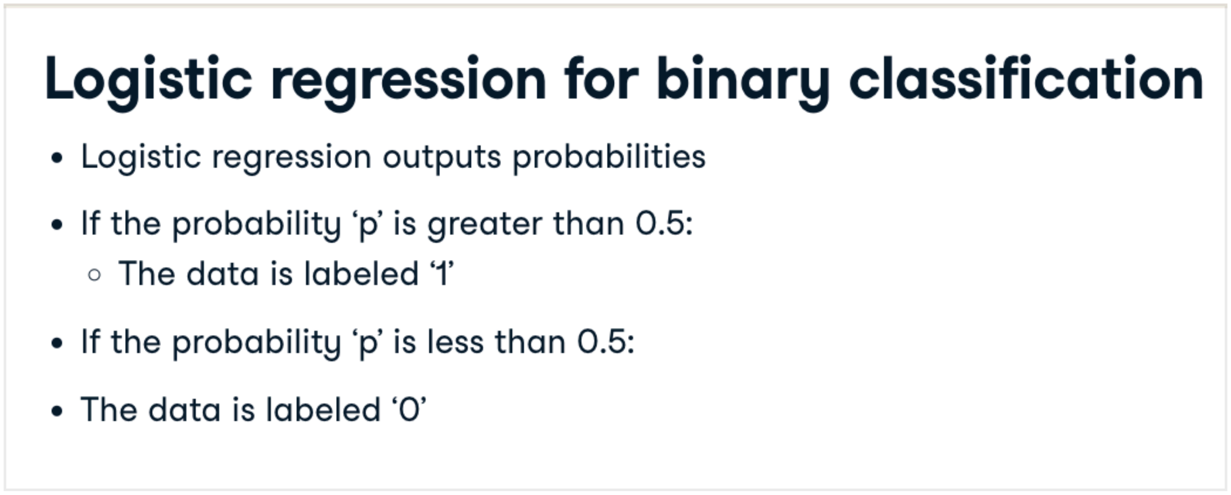
```python
print(confusion_matrix(y_test, y_pred))
```

```
[[52   7]
 [ 3 112]]
```

```python
print(classification_report(y_test, y_pred))
```

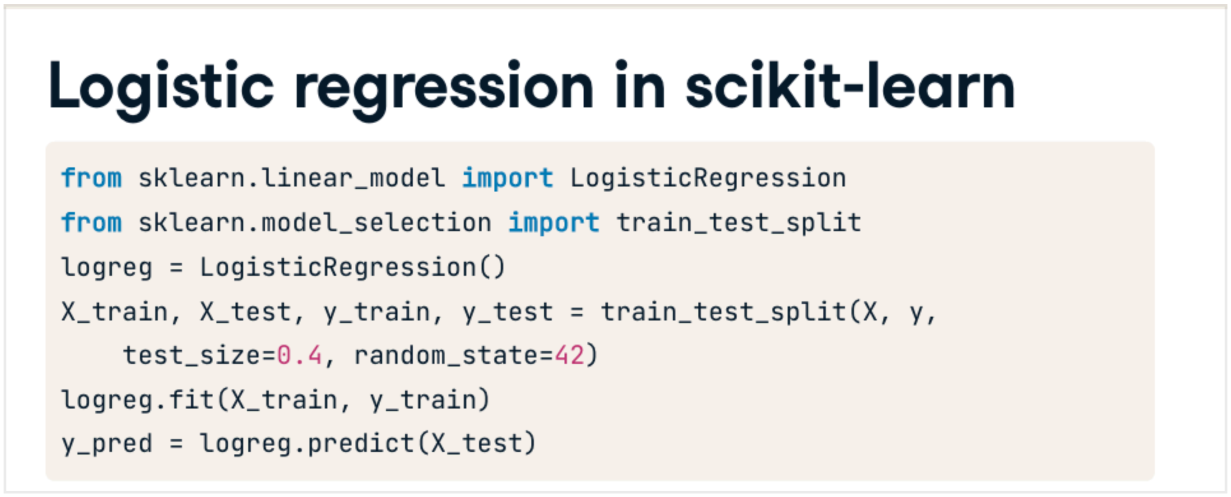
```
             precision    recall  f1-score   support
          0       0.95      0.88      0.91        59
          1       0.94      0.97      0.96       115
avg / total       0.94      0.94      0.94       174
```

Logistic regression



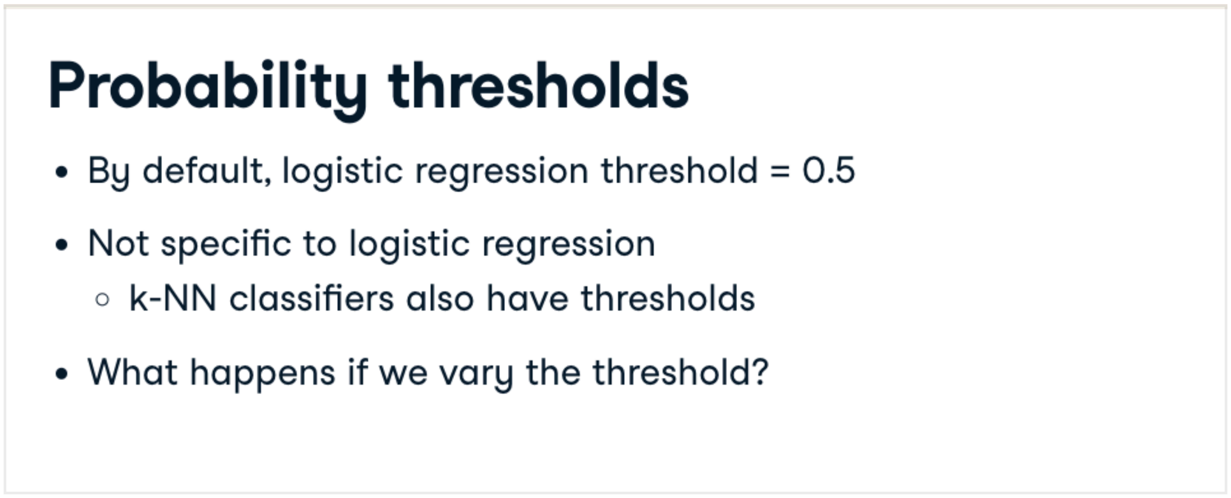
Logistic regression for binary classification

- Logistic regression outputs probabilities
- If the probability ‘p’ is greater than 0.5:
  - The data is labeled ‘1’
- If the probability ‘p’ is less than 0.5:
- The data is labeled ‘0’



Logistic regression in scikit-learn

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
```

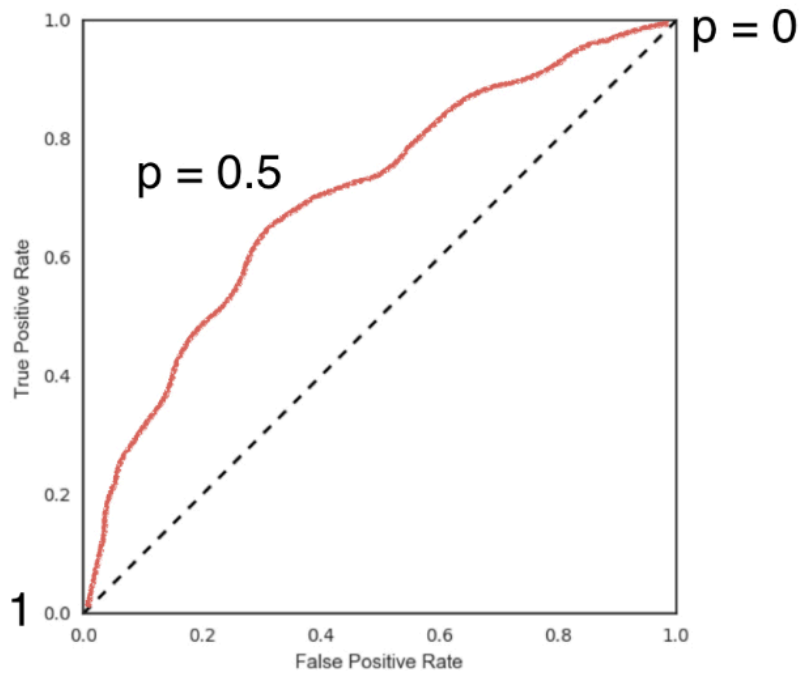


Probability thresholds

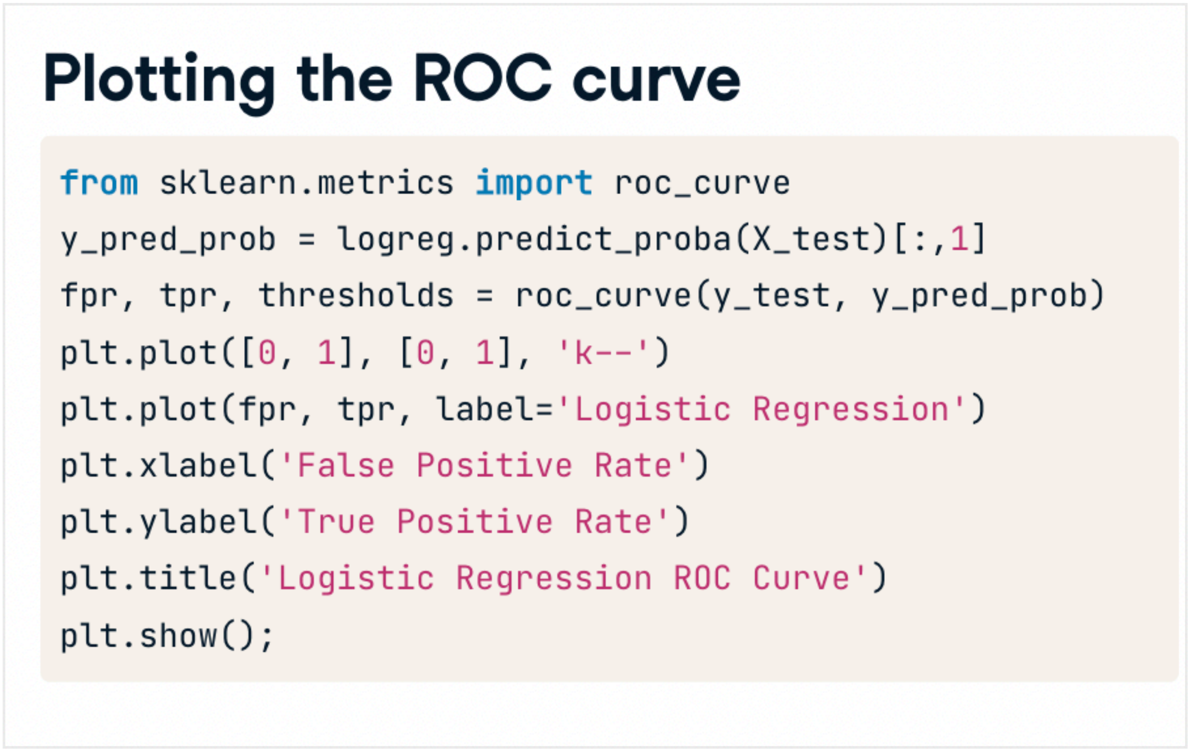
- By default, logistic regression threshold = 0.5
- Not specific to logistic regression
  - k-NN classifiers also have thresholds
- What happens if we vary the threshold?

ROC - Receiver Operating Characteristic Curve - The set of all the points that we get when trying out different thresholds is called ROC curve.

The ROC curve



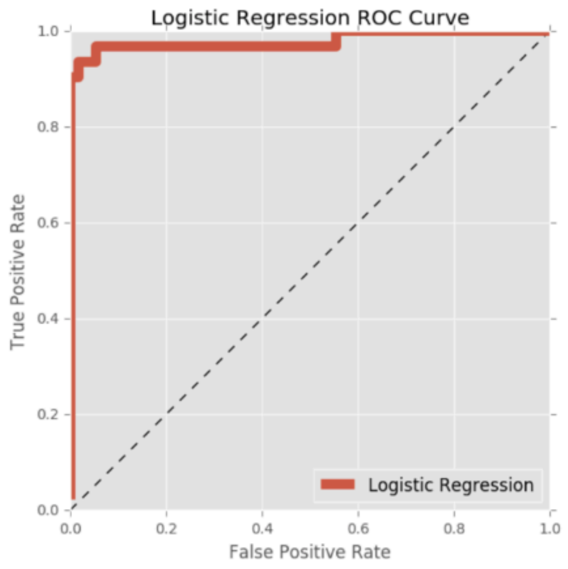
When threshold is 1, we classify all the points as 1, when threshold is 0, we classify all the points as 0.



Plotting the ROC curve

```python
from sklearn.metrics import roc_curve
y_pred_prob = logreg.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve')
plt.show();
```

Plotting the ROC curve



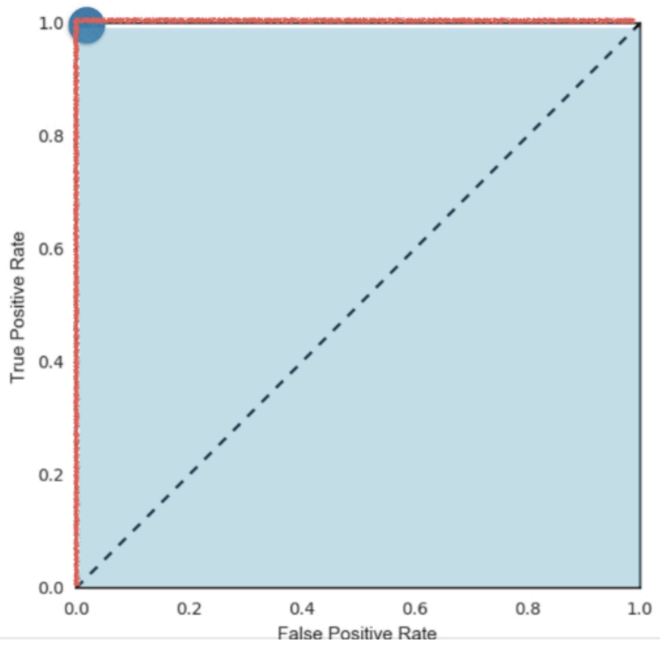
```
logreg.predict_proba(X_test)[:,1]
```

Here logreg is the model fitted on the training data
to predict proba we pass in the test data, and it outputs a 2 column data, where the first column corresponds to 1st class and the other to the 2nd class.

Area under the ROC curve (AUC)

Area under the ROC curve (AUC)

- Larger area under the ROC curve = better model

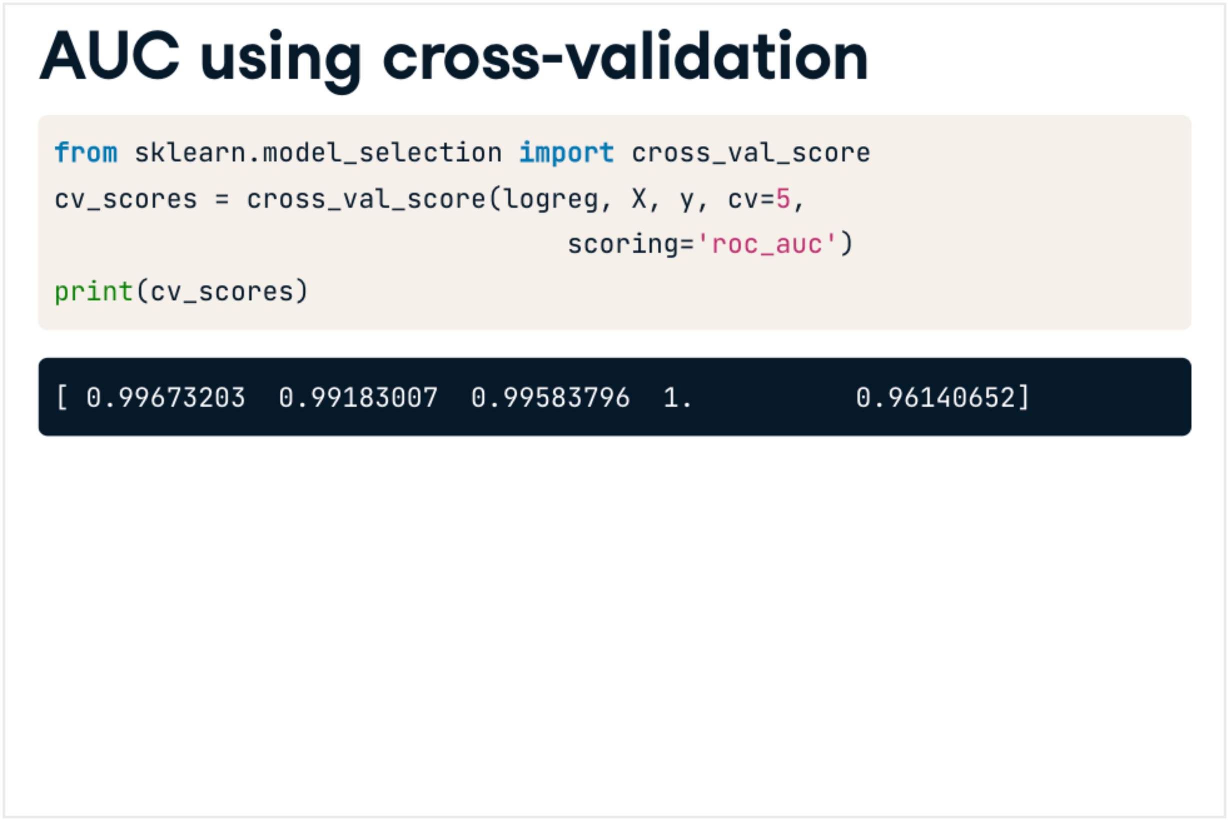


– AUC is another popular curve metric for classification models

AUC in scikit-learn

```python
from sklearn.metrics import roc_auc_score
logreg = LogisticRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=42)
logreg.fit(X_train, y_train)
y_pred_prob = logreg.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_pred_prob)
```

```
0.997466216216
```

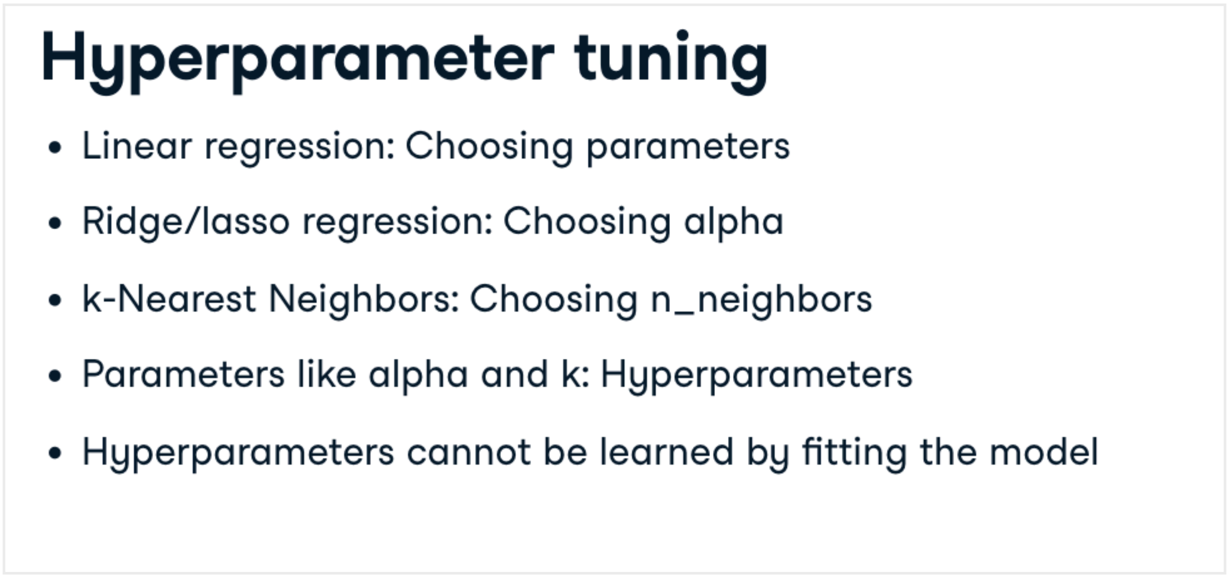


AUC using cross-validation

```python
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(logreg, X, y, cv=5,
                            scoring='roc_auc')
print(cv_scores)
```

```
[ 0.99673203  0.99183007  0.99583796  1.          0.96140652]
```

Hyperparameter Tuning



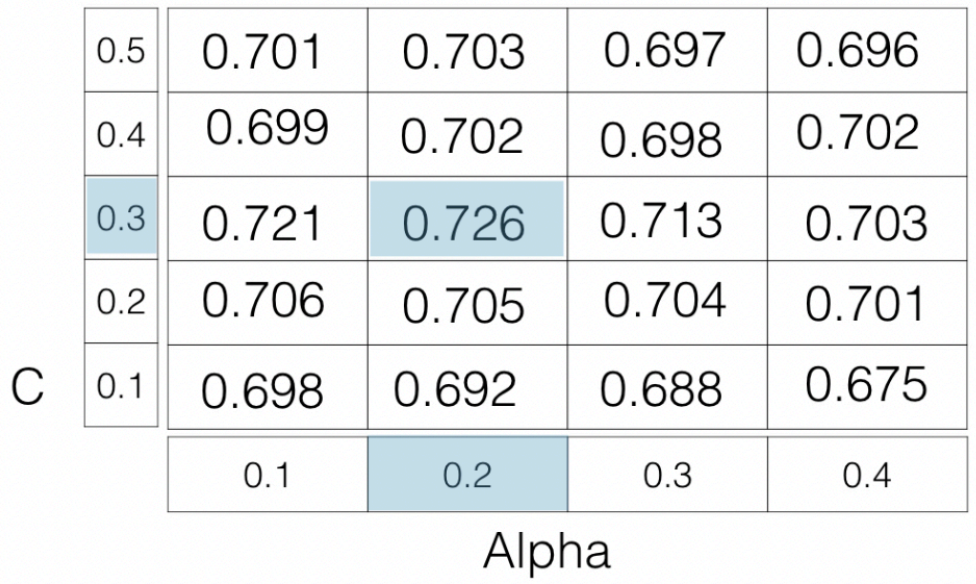
Hyperparameter tuning

- Linear regression: Choosing parameters
- Ridge/lasso regression: Choosing alpha
- k-Nearest Neighbors: Choosing n_neighbors
- Parameters like alpha and k: Hyperparameters
- Hyperparameters cannot be learned by fitting the model

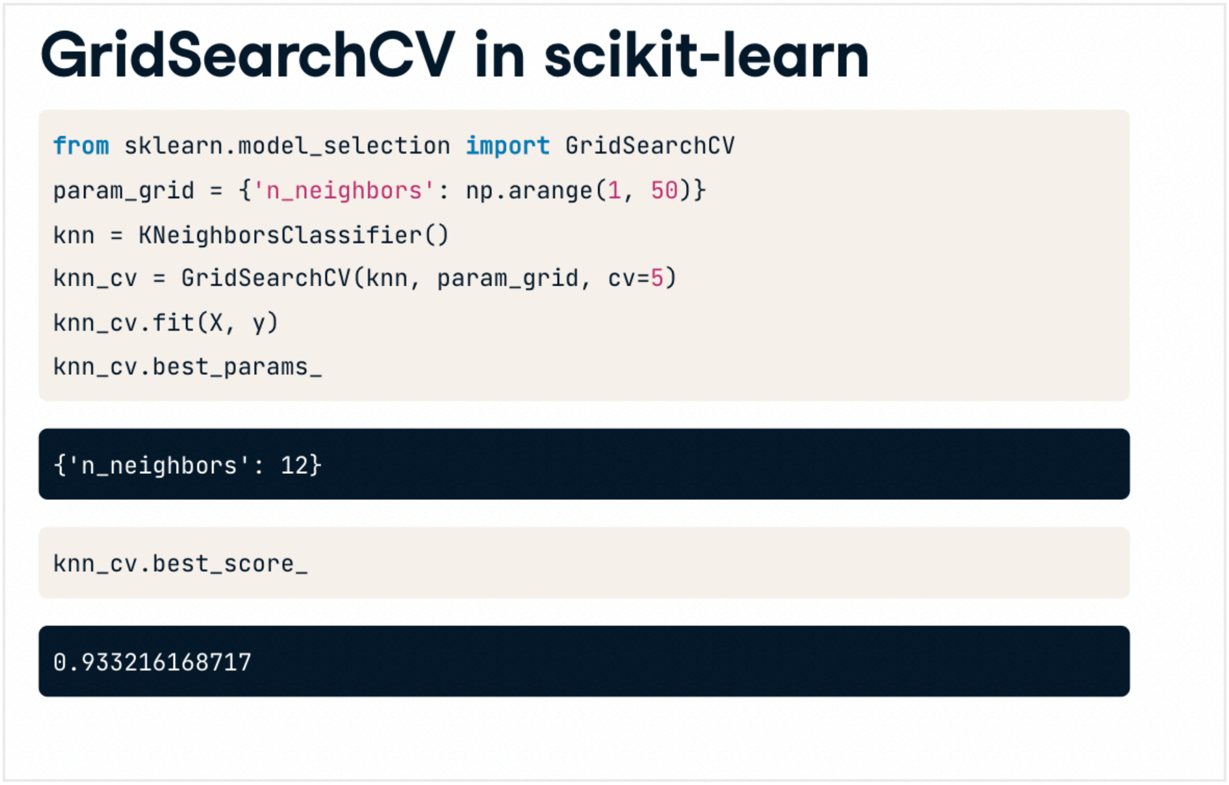
Choosing the correct hyperparameter

- Try a bunch of different hyperparameter values
- Fit all of them separately
- See how well each performs
- Choose the best performing one
- It is essential to use cross-validation

Grid search cross-validation



	0.5	0.701	0.703	0.697	0.696
	0.4	0.699	0.702	0.698	0.702
	0.3	0.721	0.726	0.713	0.703
	0.2	0.706	0.705	0.704	0.701
C	0.1	0.698	0.692	0.688	0.675
		0.1	0.2	0.3	0.4
			Alpha		



GridSearchCV in scikit-learn

```
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors': np.arange(1, 50)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid, cv=5)
knn_cv.fit(X, y)
knn_cv.best_params_
```

```
{'n_neighbors': 12}
```

```
knn_cv.best_score_
```

```
0.933216168717
```

- GridSearchCV method handles the task of selecting the best parameter by using the cross validation, to the gridsearchCV method, we pass in the parameters to tune in the form a dictionary key and the value of this key will be the values we want to perform the grid search cross validation upon.
- The best parameter will be stored in best_params_ attribute and best_score_ attribute.

Hyper-parameter tuning with RandomizedSearchCV

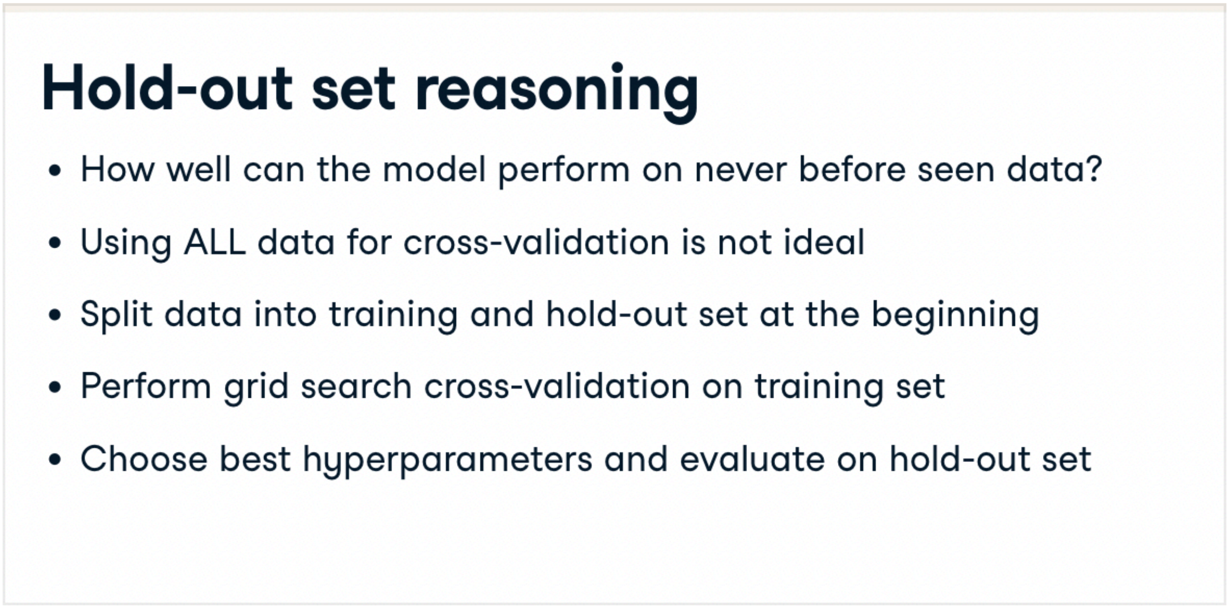
GridSearchCV can be computationally expensive, especially if you are searching over a large hyperparameter space and dealing with multiple hyperparameters. A solution to this is to use RandomizedSearchCV, in which not all hyperparameter values are tried out. Instead, a fixed number of hyperparameter settings is sampled from specified probability distributions.

Note: RandomizedSearchCV will never outperform GridSearchCV. Instead, it is valuable because it saves on computation time.

```
# Import necessary modules
from scipy.stats import randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

# Setup the parameters and distributions to sample from: param_dist
```

```
param_dist = {"max_depth": [3, None],
              "max_features": randint(1, 9),
              "min_samples_leaf": randint(1, 9),
              "criterion": ["gini", "entropy"]}

# Instantiate a Decision Tree classifier: tree
tree = DecisionTreeClassifier()

# Instantiate the RandomizedSearchCV object: tree_cv
tree_cv = RandomizedSearchCV(tree, param_dist, cv=5)

# Fit it to the data
tree_cv.fit(X, y)

# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(tree_cv.best_params_))
print("Best score is {}".format(tree_cv.best_score_))
```

Hold-out set for final evaluation



Hold-out set reasoning

- How well can the model perform on never before seen data?
- Using ALL data for cross-validation is not ideal
- Split data into training and hold-out set at the beginning
- Perform grid search cross-validation on training set
- Choose best hyperparameters and evaluate on hold-out set