

Procesamiento de Lenguaje Natural{

[NLP]

<Aprendizaje Supervisado | Grid Search &
Cross-validation>

}

```
1  La Agenda de hoy {
2
3      01    Que es Grid Search?
4
5          <Cómo funciona>
6
7          02    Que es Cross Validation?
8
9              <Cómo funciona>
10
11              03    Gridsearch & Cross
12                    Validation
13
14                  <El futuro es hoy, oiste
15                    viejo?>
16
17  }
```

1
2
3
4
5
6
7 {Grid Search}
8
9
10
11
12
13
14

Grid Search{

```
<Hasta ahora, cada que queremos probar una nueva
combinacion de hiperparametros terminamos creando un
modelo nuevo a mano>
```

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=1000, criterion='entropy')
model.fit(X_train_vectorized, y_train)
y_pred = model.predict(X_test_vectorized)

# Creamos la matriz de confusión
cm = confusion_matrix(y_test, y_pred, annot=True, cmap='BuGn', fmt='.2f')
# Imprimimos el reporte de clasificación
print(classification_report(y_test, y_pred))
```

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_jobs=-1, n_estimators=1000, criterion='entropy')
model.fit(X_train_vectorized, y_train)
y_pred = model.predict(X_test_vectorized)

# Creamos la matriz de confusión
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='BuGn', fnc='2f')
# Imprimimos el reporte de clasificación
print(classification_report(y_test, y_pred))
```

```

✓ 21.4s from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_jobs=-1, n_estimators=1000, criterion='entropy')
model.fit(X_train_vectorized, y_train)
y_pred = model.predict(X_test_vectorized)

# Creamos la matriz de confusión
sns.heatmap(
    # Imprimimos
    print(class))

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_jobs=-1, n_estimators=1000, cri
model.fit(X_train_vectorized, y_train)
y_pred = model.predict(X_test_vectorized)
✓ 21.4s

```

```

✓ 214s from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_jobs=-1, n_estimators=1000, criterion='entropy')
model.fit(X_train_vectorized, y_train)
y_pred = model.predict(X_test_vectorized)

# Creamos la matriz de confusión
sns.heatmap(
    # Imprimimos
    print(class_
weighted as
✓ 214s from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_jobs=-1, n_estimators=1000, cr
model.fit(X_train_vectorized, y_train)
y_pred = model.predict(X_test_vectorized)

```

```
# Creamos la matriz de confusión
cm = confusion_matrix(y_test, y_pred)
# Imprimimos el reporte de clasificación
print(classification_report(y_test, y_pred))

accuracy
macro avg
weighted avg
```

```
pr # Creamos la matriz de confusión
ssg.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='BuGn', fsize=24)
# Imprimimos el reporte de clasificación
print(classification_report(y_test, y_pred))

accuracy ✓ 21.4s from sklearn.ensemble import RandomForestClassifier
macro avg model: RandomForestClassifier(n_jobs=-1, n_estimators=1000, criterion=
weighted avg model.fit(X_train_vectorized, y_train)
y_pred = model.predict(X_test_vectorized)

# Creamos la matriz de confusión
```

```

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='BuGn', fsize=24)
# Imprimimos el reporte de clasificación
print(classification_report(y_test, y_pred))

```

| | precision | recall | f1-score | support |
|----------|-----------|--------|----------|---------|
| 0 | 0.95 | 0.90 | 0.93 | 6381 |
| 1 | 0.91 | 0.95 | 0.93 | 6419 |
| accuracy | | | 0.93 | 12800 |

```
fig heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='BuGn', fmt='.2f')
# Imprimamos el reporte de clasificación
print(classification_report(y_test, y_pred))
```

✓ 214s

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.95 | 0.90 | 0.93 | 6381 |
| 1 | 0.91 | 0.95 | 0.93 | 6419 |
| accuracy | | | 0.93 | 12800 |
| macro avg | 0.93 | 0.93 | 0.93 | 12800 |
| weighted avg | 0.93 | 0.93 | 0.93 | 12800 |



Grid Search{

<Y si hubiera una forma de automatizar ese proceso?>



}

```
1  Grid Search{
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

```
11
```

```
12
```

```
13
```

```
14
```

```
15
```

```
}  
}
```

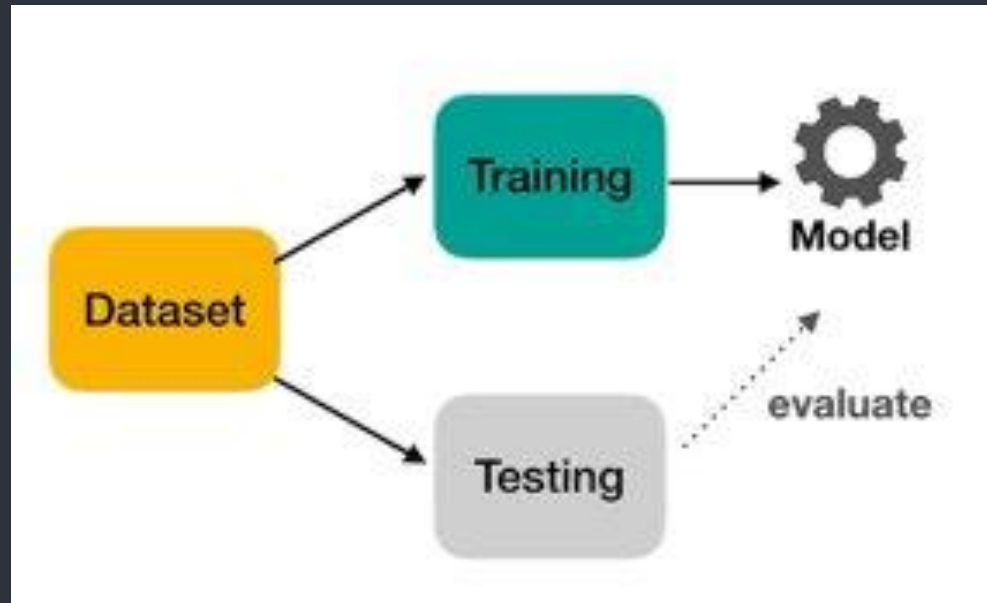
<Gridsearch realiza una busqueda exhaustiva entre las posibles combinaciones de hiperparametros para determinar que combinacion da los mejores resultados>

1
2
3
4
5
6
7
8
9
10
11
12
13
14

{Cross Validation}

Cross Validation{

<Hasta ahora, al construir modelos hacemos lo siguiente: >

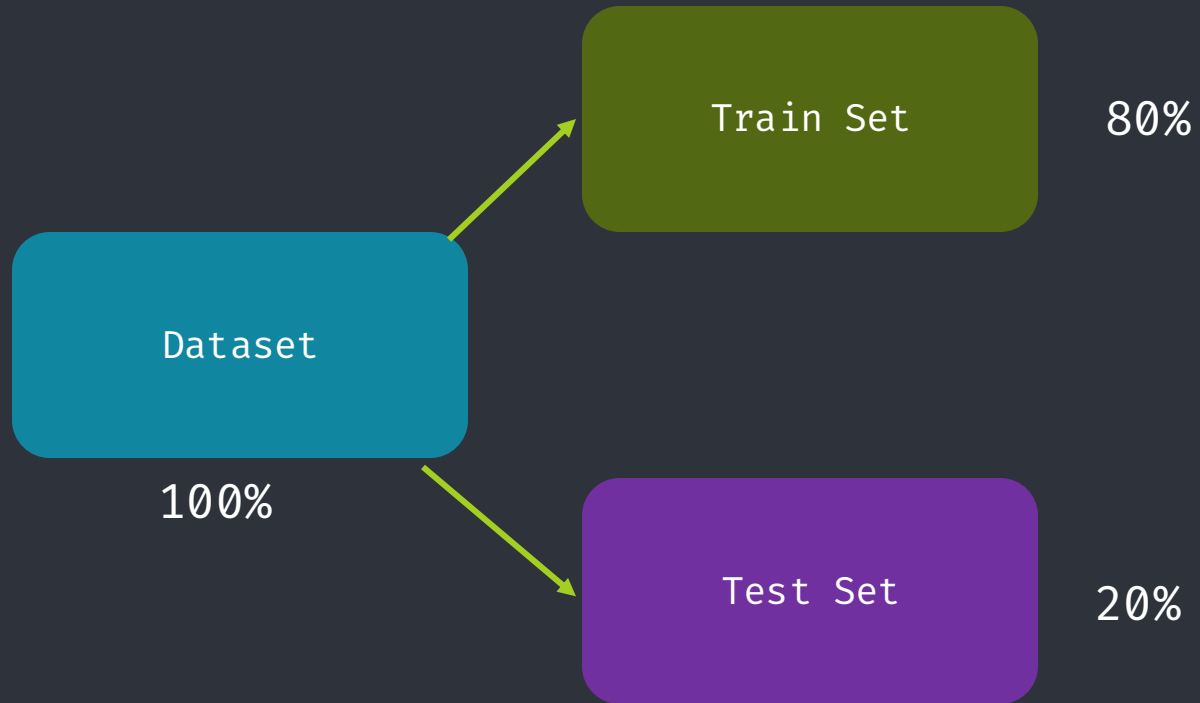


<Debemos esperar a que el modelo termine de entrenar para poder evaluar >

}

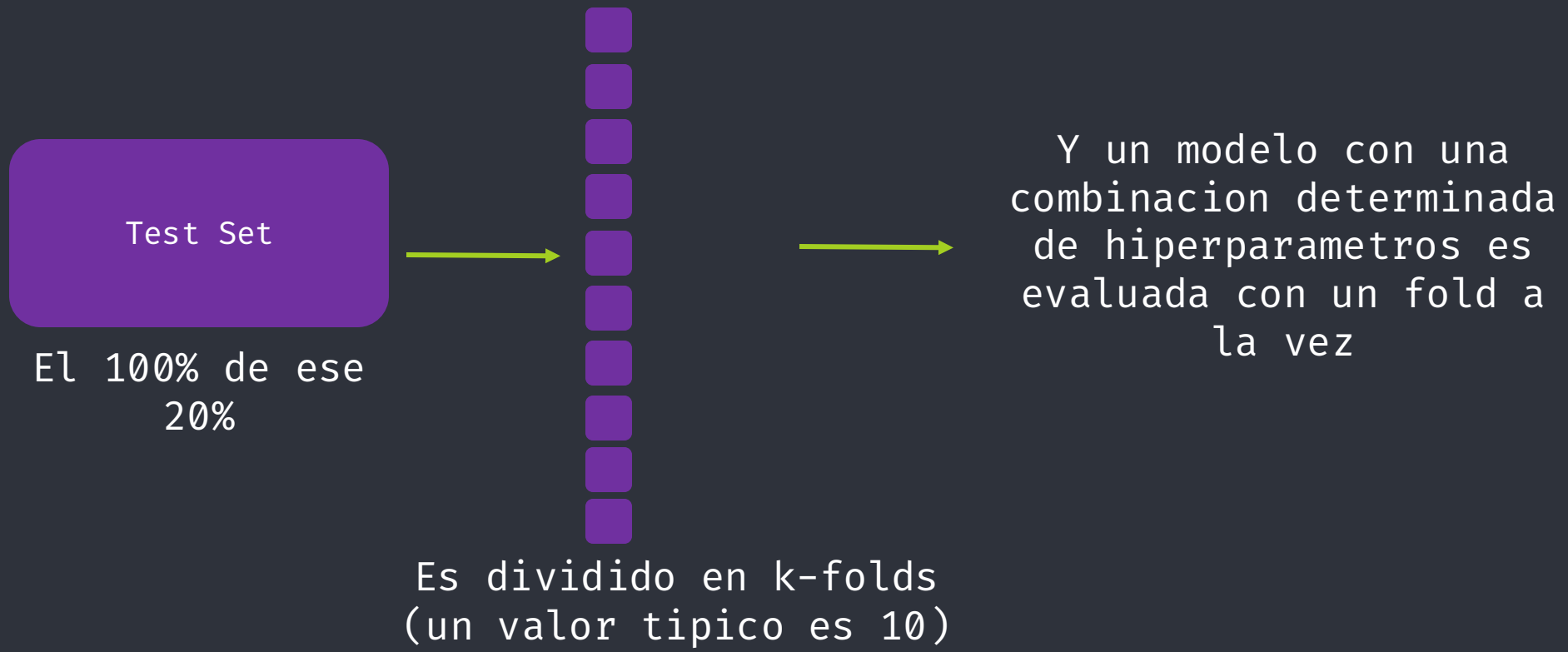
Cross Validation{

<Con cross-validation podemos evaluar desempeño durante entrenamiento>



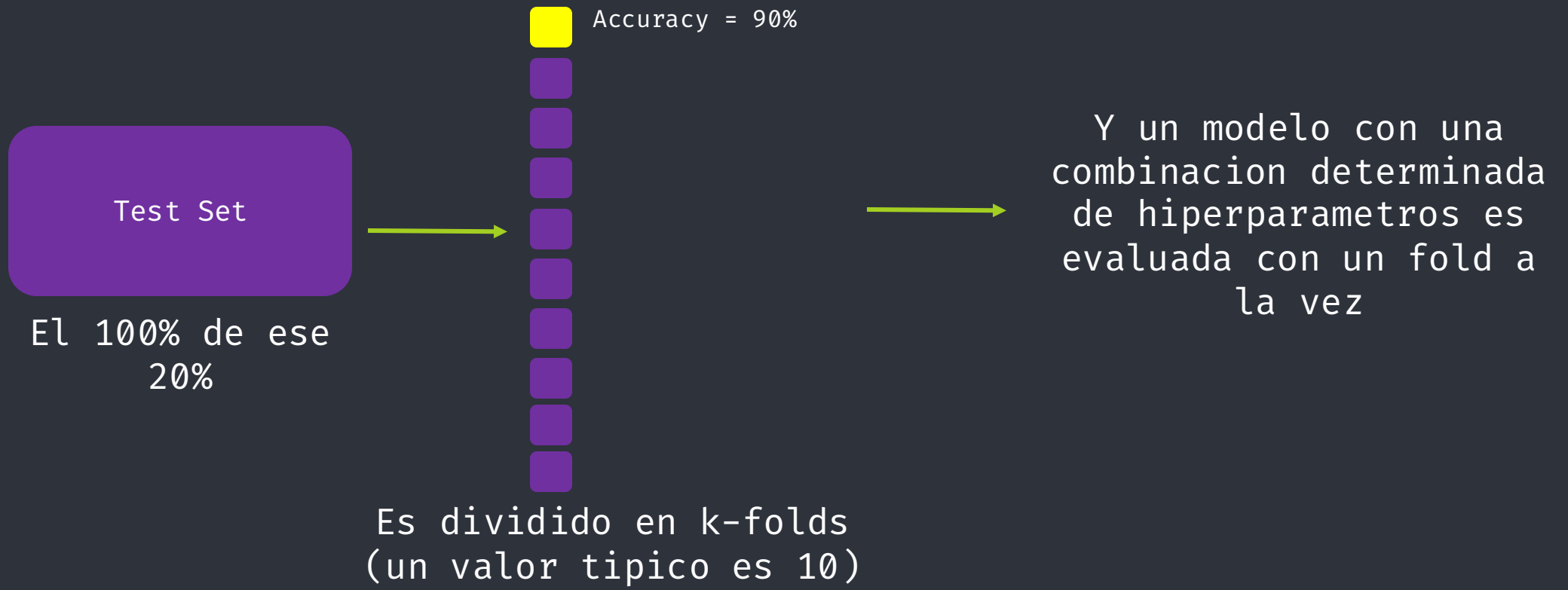
Cross Validation{

<Con cross-validation podemos evaluar desempeño durante entrenamiento>



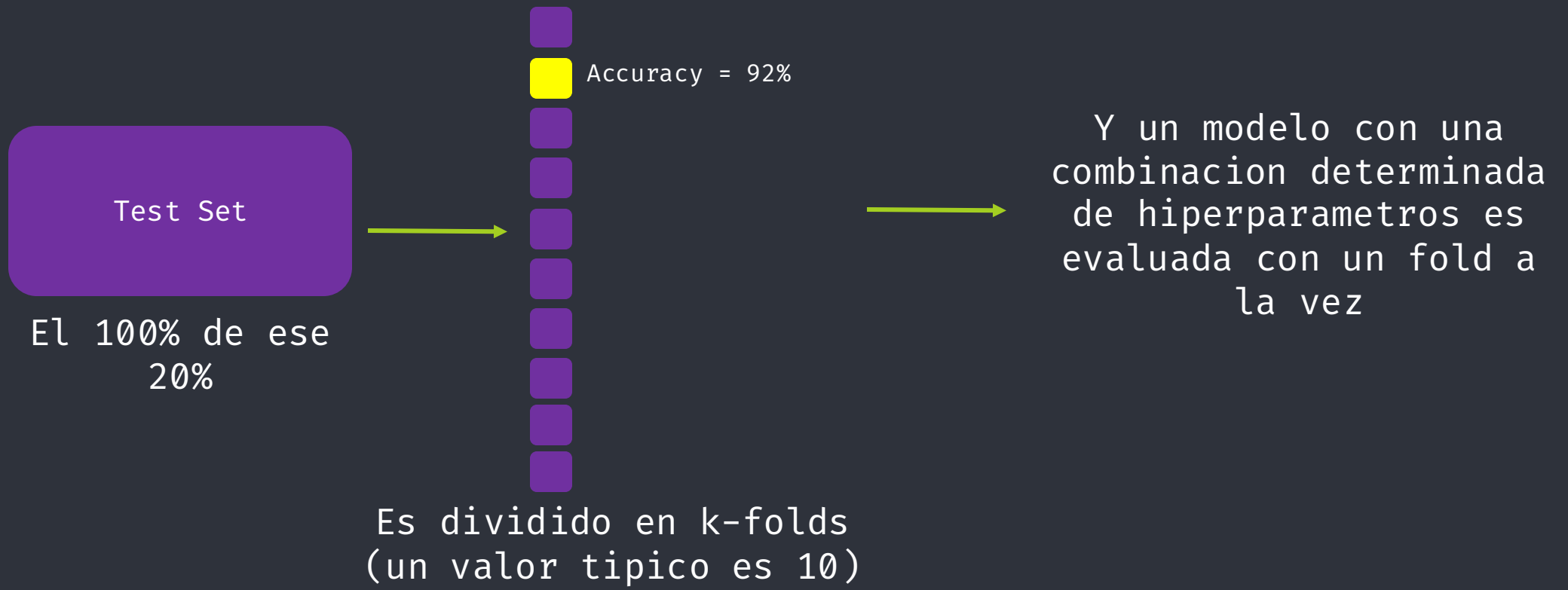
Cross Validation{

<Con cross-validation podemos evaluar desempeño durante entrenamiento>



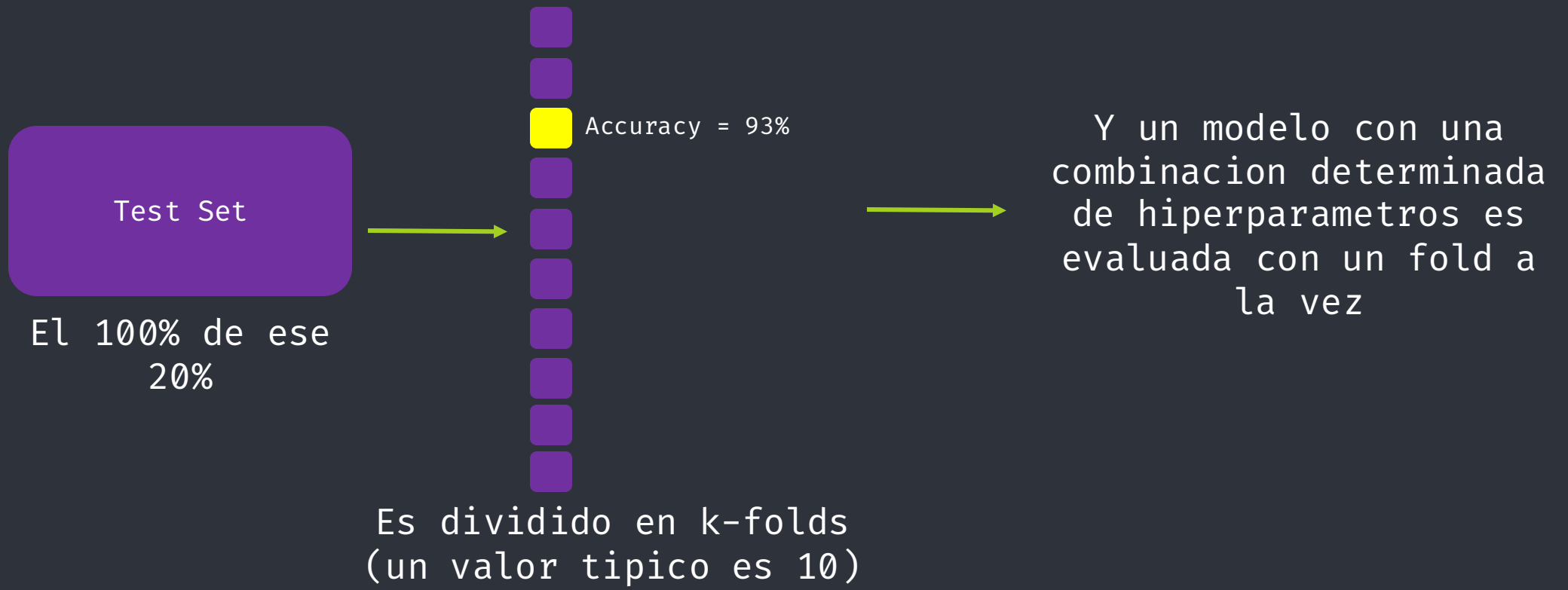
Cross Validation{

<Con cross-validation podemos evaluar desempeño durante entrenamiento>



Cross Validation{

<Con cross-validation podemos evaluar desempeño durante entrenamiento>

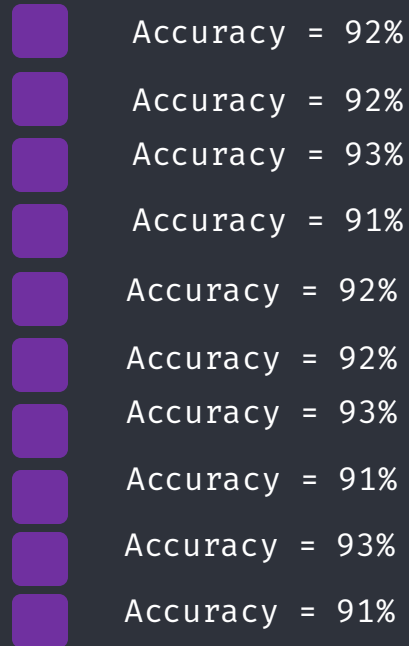


Cross Validation{

<Con cross-validation podemos evaluar desempeño durante entrenamiento>



El 100% de ese
20%



Es dividido en k-folds
(un valor tipico es 10)

Y al final obtenemos el
promedio de esas
metricas

Accuracy = 92%

```
1  
2  
3  
4 {Implementando Grid  
5 Search & Cross  
6 Validation}  
7  
8  
9  
10  
11  
12  
13  
14
```

Implementando GridSearch & Cross Validation{

<Primero importamos Pipeline y definimos un pipeline. (En NLP, estamos usando el vectorizador tfidf para preprocesar)>

```
from sklearn.pipeline import Pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', RandomForestClassifier()),
])
```

Importamos Pipeline

Paso 1 - Vectorizar

Paso 2 - Llevar esa informacion al modelo

Implementando GridSearch & Cross Validation{

<Ahora importamos GridSearchCV y definimos los parametros para testear>

```
from sklearn.model_selection import GridSearchCV
parameters = {
    'tfidf__ngram_range': [(1, 1), (1, 2)],
    'clf__n_estimators': [10, 20],
    'clf__criterion': ['gini']
}
```

Importamos GridSearchCV

Distintos ngram ranges para el vectorizador

Distinto parametros para random forest

Implementando GridSearch & Cross Validation{

<Instanciamos el grid y lo ajustamos a los datos de entrenamiento>

El numero de K-folds

-1 para paralelizar la creacion de modelos

Para imprimir en consola el progreso

```
grid = GridSearchCV(pipeline, parameters, cv=10, n_jobs=-1, verbose=3, scoring='accuracy')
grid.fit(X_train, y_train)
```

Metrica a monitorear

Estimador. En este caso el Pipeline

Los parametros para testear

Implementando GridSearch & Cross Validation{

<Y ahora esperamos>



Implementando GridSearch & Cross Validation{

<Una vez el proceso termine, podemos obtener el mejor estimador y sus metricas haciendo lo siguiente>

```
grid.best_params_
```

✓ 0.0s

```
{'clf__criterion': 'gini',  
 'clf__n_estimators': 20,  
 'tfidf__ngram_range': (1, 1)}
```

Combinacion de parametros que
entregó el accuracy mas alto

```
grid.best_score_
```

✓ 0.0s

0.9153515624999999

Accuracy score del mejor estimador

1
2
3
4
5
6
7
8
9
1
0
1
2
3
4

Q&A