03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

1 MACHINE LEARNING – HIPERPARÁMETROS

Para entender qué son los hiperparámetros, vamos a ver la diferencia entre parámetros e hiperparámetros:

Diferencia entre parámetro e hiperparámetro:

- y = mx + n -> m,n son parámetros; **valores que queremos obtener durante el entrenamiento**
- `clf = svm.SVC(kernel='linear', C=1)`; **kernel, c** son hiperparámetros; valores que nos ayudan a optmizar nuestra función a la hora de entrenar.

Dependiendo de los valores de nuestros hiperparámetros, nuestro modelo será capaz de obtener unos parámetros que se ajusten mejor o peor a los datos. Veamos un ejemplo:

```
[ ]  clf = svm.SVC(C=1).fit(X_train, y_train) # c: parámetro de regularización
     clf.score(X_test, y_test)

     0.9333333333333333
```

```
[ ]  clf = svm.SVC(C=6).fit(X_train, y_train)
     clf.score(X_test, y_test)

     0.9666666666666667
```

Dependiendo del valor de C, incrementamos o decrementamos el score.

Por lo tanto, la duda que nos surge es: **¿Cómo se cuáles son los mejores hiperparámetros?**

Hay distintas metodologías que nos permiten encontrar los valores más óptimos.

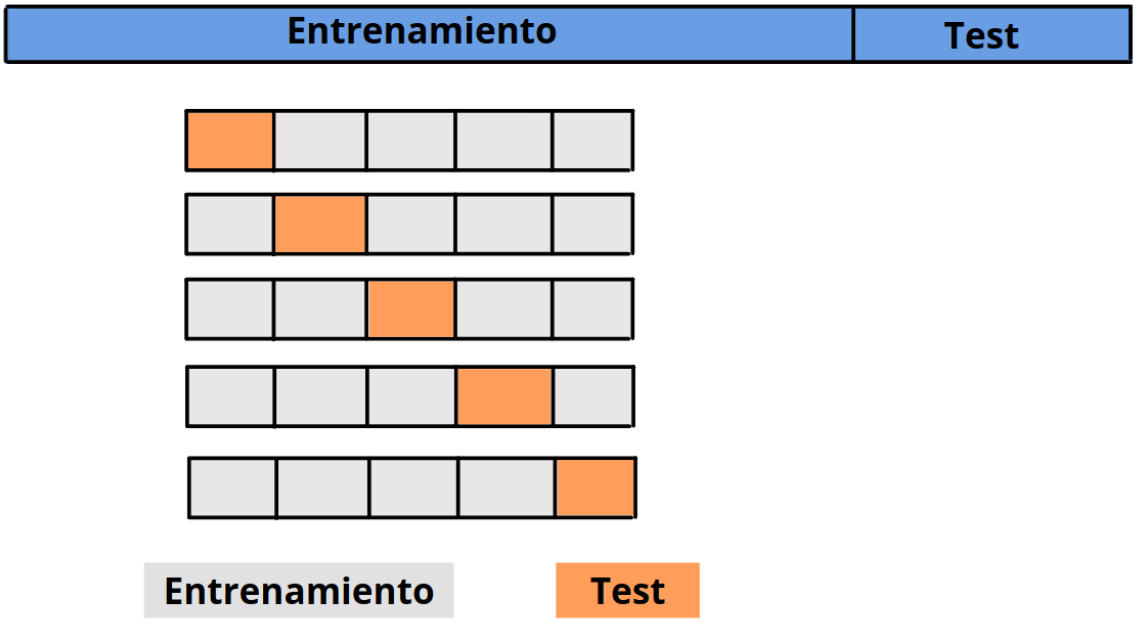
Dependiendo del problema al que nos estemos enfrentando, quizá nos interesa obtener una mayor accuracy, o quizá una mejor precisión. En cualquier caso, lo más probable es que para obtener una métrica u otra no se obtengan los mismos hiperparámetros.

Muchas veces el resultado depende de cómo se han elegido los valores de entrenamiento y los valores de test.

Para que el resultado no dependa tanto de estos datos, se aplica una técnica de separación de datos en train y test que se denomina

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

K-FOLD CROSS VALIDATION



Basados en el dataset de Flores IRIS:

1. Separamos como siempre el train y el test

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.3)
```

2. Aplicamos el algoritmo SVM

```
model = svm.SVC(kernel='rbf',C=30,gamma='auto')
model.fit(X_train,y_train)
model.score(X_test, y_test)
```

0.911111

3. Usamos K-Fold Validation
   Aplicamos MANUALMENTE el algoritmo SVM con 5 conjuntos de datos train y test:

```
: cross_val_score(svm.SVC(kernel='linear',C=10,gamma='auto'),iris.data, iris.target, cv=5)
: array([1.        , 1.        , 0.9       , 0.96666667, 1.        ])

: cross_val_score(svm.SVC(kernel='rbf',C=10,gamma='auto'),iris.data, iris.target, cv=5)
: array([0.96666667, 1.        , 0.96666667, 0.96666667, 1.        ])

: cross_val_score(svm.SVC(kernel='rbf',C=20,gamma='auto'),iris.data, iris.target, cv=5)
: array([0.96666667, 1.        , 0.9       , 0.96666667, 1.        ])
```

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

4. Aproximación con un bucle para no hacerlo manual:

```
kernels = ['rbf', 'linear']
C = [1,10,20]
avg_scores = {}
for kval in kernels:
    for cval in C:
        cv_scores =
cross_val_score(svm.SVC(kernel=kval,C=cval,gamma='auto'),iris.data, iris.target,
cv=5)
        avg_scores[kval + '_' + str(cval)] = np.average(cv_scores)

avg_scores
```

SALIDA:

```
{'rbf_1': 0.9800000000000001,
 'rbf_10': 0.9800000000000001,
 'rbf_20': 0.9666666666666668,
 'linear_1': 0.9800000000000001,
 'linear_10': 0.9733333333333334,
 'linear_20': 0.9666666666666666}
```

5. Usando GRIDSearchCV

```
from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(svm.SVC(gamma='auto'), {
    'C': [1,10,20],
    'kernel': ['rbf','linear']
}, cv=5, return_train_score=False)
clf.fit(iris.data, iris.target)
clf.cv_results_
df = pd.DataFrame(clf.cv_results_)
df
```

```
df[['param_C','param_kernel','mean_test_score']]
```

	param_C	param_kernel	mean_test_score
0	1	rbf	0.980000
1	1	linear	0.980000
2	10	rbf	0.980000
3	10	linear	0.973333
4	20	rbf	0.966667
5	20	linear	0.966667

```
clf.best_params_
```

```
{'C': 1, 'kernel': 'rbf'}
```

```
clf.best_score_
```

```
0.98
```

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

También se usa **RandomizedSearchCV** para reducir el número de iteraciones y con una combinación aleatoria de parámetros. Esto resulta útil cuando hay que probar demasiados parámetros y el tiempo de entrenamiento es mayor. Ayuda a reducir el coste de computación.

```
from sklearn.model_selection import RandomizedSearchCV
rs = RandomizedSearchCV(svm.SVC(gamma='auto'), {
        'C': [1,10,20],
        'kernel': ['rbf','linear']
    },
    cv=5,
    return_train_score=False,
    n_iter=2
)
rs.fit(iris.data, iris.target)
pd.DataFrame(rs.cv_results_)[['param_C','param_kernel','mean_test_score']]
```

	param_C	param_kernel	mean_test_score
0	10	rbf	0.98
1	1	linear	0.98

Pero también podemos usar diferentes modelos con diferentes hiperparámetros:

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

model_params = {
    'svm': {
        'model': svm.SVC(gamma='auto'),
        'params' : {
            'C': [1,10,20],
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(),
        'params' : {
            'n_estimators': [1,5,10]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(solver='liblinear',multi_class='auto'),
        'params': {
            'C': [1,5,10]
        }
    }
}
```

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

```
scores = []

for model_name, mp in model_params.items():
    clf =  GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
    clf.fit(iris.data, iris.target)
    scores.append({
        'model': model_name,
        'best_score': clf.best_score_,
        'best_params': clf.best_params_
    })

df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

	model	best_score	best_params
0	svm	0.980000	{'C': 1, 'kernel': 'rbf'}
1	random_forest	0.953333	{'n_estimators': 5}
2	logistic_regression	0.966667	{'C': 5}

Por tanto podemos concluir que para el dataset de FLORES IRIS el mejor algoritmo es el SVM con hiperparámetros C: 1 y kernel: 'rbf'