Badr TADJER | Naoufal ARRADI | Leo TRAN
LS1-M1-APP

Session 3 – TP Machine Learning

Model training

Some of the columns don't have predictive power, so let's specify which ones are included for prediction

Entrée [13]:
```
predictors = ["Pclass", "Sex", "Age", 'SibSp', 'Parch', "Fare", "Embarked"]
```

We need now to convert text columns in **predictors** to numerical ones

Entrée [14]:
```
for col in predictors: # Loop through all columns in predictors
    if titanic[col].dtype == 'object':  # check if column's type is object (text)
        titanic[col] = pd.Categorical(titanic[col]).codes  # convert text to numerical

titanic.head()
```

Out[14]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	2
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	0
3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	2
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	2
5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	2

Entrée [27]:
```
# Split the data into a training set and a testing set. Set: test_size=0.3, random_state=1
from sklearn.model_selection import train_test_split
# your code here

y = titanic["Survived"]
X = titanic[predictors]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)

print ("train shape", X_train.shape, y_train.shape)
print ("test shape", X_test.shape, y_test.shape)
```

```
train shape (623, 7) (623,)
test shape (268, 7) (268,)
```

Entrée [28]:
```
# import LogisticRegression from: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegr

# your code here
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state = 1)
clf = clf.fit(X_train, y_train)

# your code here
train_score = clf.score(X_train, y_train)
test_score = clf.score(X_test, y_test)

print ('train accuracy =', train_score)
print ('test accuracy =', test_score)
```

```
train accuracy = 0.8073836276083467
test accuracy = 0.7723880597014925
```

Train accuracy indique que 80% de nos trains sont corrects et Test accuracy indique que 77% de nos tests sont corrects.
En conclusion, les valeurs sont assez proches, par conséquent nous ne sommes pas sur un cas d'overfitting.

Let's print the model's parameters

Entrée [17]:
```
coeff = pd.DataFrame()
coeff['Feature'] = X_train.columns
coeff['Coefficient Estimate'] = pd.Series(clf.coef_[0])
coeff.loc[len(coeff)]=['Intercept', clf.intercept_[0]]
print(coeff)
```

```
     Feature  Coefficient Estimate
0     Pclass             -1.158693
1        Sex             -2.708761
2        Age             -0.040634
3      SibSp             -0.334012
4      Parch              0.071939
5       Fare             -0.000570
6   Embarked             -0.223307
7  Intercept              5.391545
```

We now need to predict class labels for the test set. We will also generate the class probabilities

Entrée [18]:
```
# predict class labels for the test set
y_pred = clf.predict(X_test)
print (y_pred)
```

```
[1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 1 0 1 0
 0 1 0 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0
 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0
 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 0 0 1
 0 0 1 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0
 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0
 1 1 0 1 1 0 0 1 0]
```

Entrée [19]:
```
# generate class probabilities : http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegre
y_probs = clf.predict_proba(X_test)
print (y_probs)
```

```
[[0.13931138 0.86068862]
 [0.91545699 0.08454301]
 [0.12666421 0.87333579]
 [0.36591842 0.63408158]
 [0.07784301 0.92215699]
 [0.89067951 0.10932049]
 [0.80772606 0.19227394]
 [0.11922783 0.88077217]
 [0.49862954 0.50137046]
 [0.44730682 0.55269318]
 [0.90553194 0.09446806]
 [0.39942792 0.60057208]
 [0.75883959 0.24116041]
 [0.7731844  0.2268156 ]
 [0.36586717 0.63413283]
 [0.51884413 0.48115587]
 [0.92555342 0.07444658]
 [0.92813017 0.07186983]
 [0.93745225 0.06254775]
```

Nous avons ici la probabilité de mort à gauche et la probabilité de survie à droite.
Afin de choisir si le résultat sera Mort ou Survie, la machine sélectionne la probabilité la plus haute entre les deux.

As you can see, the classifier outputs two probabilities for each row. It's predicting a 1 (Survived) any time the probability in the second column is greater than 0.5. Let's visualize it all together.

Entrée [20]:
```
pred = pd.DataFrame({
        "Survived_original": y_test,
        "Survived_predicted": y_pred,
        "Survived_proba": np.transpose(y_probs)[1]
        })
pred["Comparison"] = pred.Survived_original == pred.Survived_predicted
pred.head()
```

Out[20]:

PassengerId	Survived_original	Survived_predicted	Survived_proba	Comparison
863	1	1	0.860689	True
224	0	0	0.084543	True
85	1	1	0.873336	True
681	0	1	0.634082	False
536	1	1	0.922157	True

Confusion matrix

Entrée [21]:
```
from sklearn import metrics
print (metrics.confusion_matrix(y_test, y_pred))
print (metrics.classification_report(y_test, y_pred))
```

```
[[129  24]
 [ 37  78]]
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       153
           1       0.76      0.68      0.72       115

    accuracy                           0.77       268
   macro avg       0.77      0.76      0.76       268
weighted avg       0.77      0.77      0.77       268
```

As you can see, we can have the classification report for each class

Notre matrice de confusion est la suivante :

		Predicted class	
		Negative	**Positive**
Actual class	**Negative**	**129**	**24**
	Positive	**37**	**78**

K-Fold Cross Validation

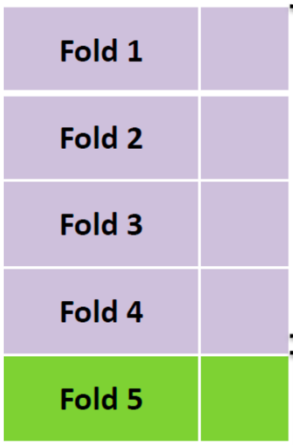
Entrée [24]:
```
# import cross_validation from: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_s
# your code here
from sklearn.model_selection import cross_val_score
clf = LogisticRegression(random_state = 1)
scores = cross_val_score(clf, titanic[predictors], titanic["Survived"], scoring = 'accuracy', cv = 5)
## see model
print(scores)
# Take the mean of the scores (because we have one for each fold)
print(scores.mean())
```

```
[0.7877095  0.78651685 0.78089888 0.76966292 0.82022472]
0.7890025735986442
```

When you are improving a model, you want to make sur that you are really doing it and not just being lucky. This is why it's good to work with cross validation instead of one train/test split.

Entrée [ ]:

Ici nous avons divisé notre modèle en 5 Folds et nous avons réitéré l'opération 5 fois.



Nous avons obtenu les précisions suivantes :
[0.7877095, 0.78651685, 0.78089888, 0.76966292, 0.82022472]
Et la moyenne des ces précisions est de **0.7890025735986442,** donc 78 % d'exactitude en moyenne.