

예제 : <https://docs.microsoft.com/ko-kr/azure/hdinsight/spark/apache-spark-machine-learning-mllib-ipython>

Spark MLlib Machine Learning Appliacation

데이터랩 3반 정미연

이 예제에서는 [시카고 데이터 포털](https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5)을 통해 획득한
식품 검사 데이터(**Food_Inspections1.csv**)에 대한
예측 분석을 다룹니다.

<https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5>

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row
from pyspark.sql.functions import UserDefinedFunction
from pyspark.sql.types import *
```

파이스파크 라이브러리들을 불러옵니다.

Pipeline : 파이프라인 (프로세스)를 그려줌

Logistic Regression : 로지스틱 회귀분석 모듈

HashingTF : 문서의 갯수를 해싱함

Tokenizer : 토큰화

Row : DataFrame 형태에서의 행

UserDefinedFunction : to define new [Column](#)-based functions that extend the vocabulary of Spark SQL's DSL for transforming [Datasets](#).

해시함수(hash function)란 데이터의 효율적 관리를 목적으로 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 함수입니다. 이 때 매핑 전 원래 데이터의 값을 **키(key)**, 매핑 후 데이터의 값을 **해시값(hash value)**, 매핑하는 과정 자체를 **해싱(hashing)**라고 합니다.

원시 데이터가 CSV 형식이기 때문에 Spark 컨텍스트를 사용하여 파일을 메모리에 구조화되지 않은 데이터로 가져온 다음, Python의 CSV 라이브러리를 사용하여 데이터의 각 줄을 구문 분석할 수 있습니다.

```
def csvParse(s):  
    import csv  
    from StringIO import StringIO  
    sio = StringIO(s)  
    value = csv.reader(sio).next()  
    sio.close()  
    return value  
  
inspections = sc.textFile('wasb:///HdiSamples/HdiSamples/FoodInspectionData/Food_Inspections1.csv')\  
                .map(csvParse)
```

입력 데이터를 가져오고 구문 분석하여 RDD(복원 분산 데이터 집합)를 만듭니다.

```
inspections.take(1)
```

RDD에서 한 행을 검색하면 데이터 스키마의 모양을 살펴볼 수 있습니다.

```
[['413707',  
  'LUNA PARK INC',  
  'LUNA PARK DAY CARE',  
  '2049789',  
  "Children's Services Facility",  
  'Risk 1 (High)',  
  '3250 W FOSTER AVE ',  
  'CHICAGO',  
  'IL',  
  '60625',  
  '09/21/2010',  
  'License-Task Force',  
  'Fail',  
  '24. DISH WASHING FACILITIES: PROPERLY DESIGNED, CONSTRUCTED, MAINTAINED, INSTALLED, LOCATED AND',  
  '41.97583445690982',  
  '-87.7107455232781',  
  '(41.97583445690982, -87.7107455232781)']]
```

모든 회사의 이름, 회사의 종류, 주소, 검사 데이터 및 위치가 포함되어 있습니다.

다음 코드를 실행하여 예측 분석에 유용한 몇 개 열이 포함된 데이터 프레임(*df*) 및 임시 테이블 (*CountResults*)을 만듭니다. *sqlContext*는 구조화된 데이터에서 변환을 수행하는 데 사용할 수 있습니다.

```
schema = StructType([
    StructField("id", IntegerType(), False),
    StructField("name", StringType(), False),
    StructField("results", StringType(), False),
    StructField("violations", StringType(), True)])
```

```
df = spark.createDataFrame(inspections.map(lambda l: (int(l[0]), l[1], l[12], l[13])) ,
schema)
df.registerTempTable('CountResults')
```

데이터 프레임에서 관심이 있는 네 개의 열은 **id**, **name**, **results** 및 **violations**입니다.

```
df.show(5)
```

아이디와 이름, 결과, 위반사항을 확인합니다.

id	name	results	violations
413707	LUNA PARK INC	Fail	24. DISH WASHING ...
391234	CAFE SELMARIE	Fail	2. FACILITIES TO ...
413751	MANCHU WOK	Pass	33. FOOD AND NON-...
413708	BENCHMARK HOSPITA...	Pass	
413722	JJ BURGER	Pass	

```
df.select('results').distinct().show()
```

결과값의 고유 값을 확인합니다.

```
+-----+  
|          results|  
+-----+  
|              Fail|  
|Business Not Located|  
|              Pass|  
|  Pass w/ Conditions|  
|      Out of Business|  
+-----+
```

결과값의 분포 값을 확인합니다.

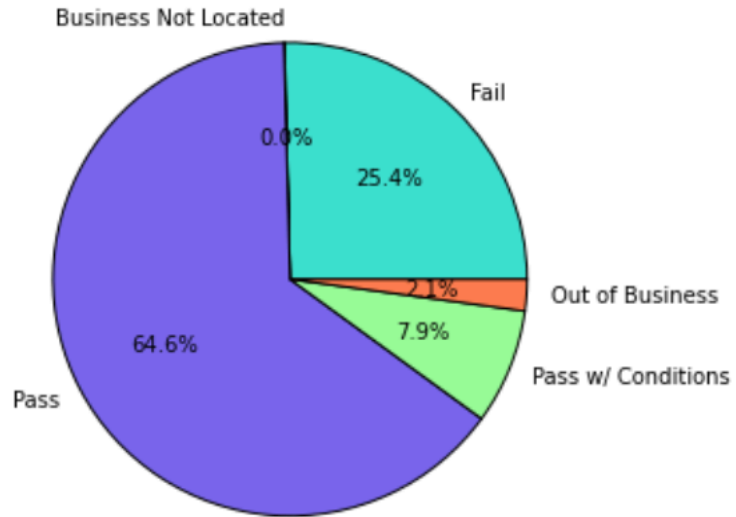
```
%%sql -o countResultsdf  
SELECT results, COUNT(results) AS cnt FROM CountResults GROUP BY results
```

cnt	results
3324	Fail
6	Business Not Located
8457	Pass
1033	Pass w/ Conditions
281	Out of Business

결과를 시각화 해봅니다.

```
%%local
%matplotlib inline
import matplotlib.pyplot as plt

labels = countResultsdf['results']
sizes = countResultsdf['cnt']
colors = ['turquoise', 'seagreen', 'mediumslateblue', 'palegreen', 'coral']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors)
plt.axis('equal')
```



회사를 찾을 수 없음 0%
불합격 25.4%
합격 64.6%
조건부 합격 7.9%
폐업 7.1%

로지스틱 회귀분석은

종속변수가 이진(1, 0)으로 되어있는 분류기

합격(1)

- 합격
- 조건부 합격

불합격(0)

- 불합격

취소(-1)

- 회사를 찾을 수 없음
- 폐업

<결과 비율>

회사를 찾을 수 없음 0%

불합격 25.4%

합격 64.6%

조건부 합격 7.9%

폐업 7.1%

앞에서 언급한 대로 로지스틱 회귀분석 labelForResults를 생성합니다.

```
def labelForResults(s):  
    if s == 'Fail':  
        return 0.0  
    elif s == 'Pass w/ Conditions' or s == 'Pass':  
        return 1.0  
    else:  
        return -1.0  
label = UserDefinedFunction(labelForResults, DoubleType())  
labeledData = df.select(label(df.results).alias('label'), df.violations).where('label >= 0')
```

레이블이 지정된 데이터의 한 행을 표시합니다.

```
labeledData.take(1)
```

1.[Row(label=0.0, violations=u"41. PREMISES MAINTAINED FREE OF LITTER, UNNECESSARY ARTICLES, CLEANING EQUIPMENT PROPERLY STORED - Comments: All parts of the food establishment and all parts of the property used in connection with the operation of the establishment shall be kept neat and clean and should not produce any offensive odors. REMOVE MATTRESS FROM SMALL DUMPSTER. | 35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: The walls and ceilings shall be in good repair and easily cleaned. REPAIR MISALIGNED DOORS AND DOOR NEAR ELEVATOR. DETAIL CLEAN BLACK MOLD LIKE SUBSTANCE FROM WALLS BY BOTH DISH MACHINES. REPAIR OR REMOVE BASEBOARD UNDER DISH MACHINE (LEFT REAR KITCHEN). SEAL ALL GAPS. REPLACE MILK CRATES USED IN WALK IN COOLERS AND STORAGE AREAS WITH PROPER SHELVING AT LEAST 6' OFF THE FLOOR. | 38. VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED AND MAINTAINED - Comments: The flow of air discharged from kitchen fans shall always be through a duct to a point above the roofline. REPAIR BROKEN VENTILATION IN MEN'S AND WOMEN'S WASHROOMS NEXT TO DINING AREA. | 32. FOOD AND NON-FOOD CONTACT SURFACES PROPERLY DESIGNED, CONSTRUCTED AND MAINTAINED - Comments: All food and non-food contact equipment and utensils shall be smooth, easily cleanable, and durable, and shall be in good repair. REPAIR DAMAGED PLUG ON LEFT SIDE OF 2 COMPARTMENT SINK. REPAIR SELF CLOSER ON BOTTOM LEFT DOOR OF 4 DOOR PREP UNIT NEXT TO OFFICE.")]

마지막 작업은 레이블이 지정된 데이터를 로지스틱 회귀로 분석할 수 있는 형식으로 변환하는 것입니다. 로지스틱 회귀 알고리즘에 대한 입력은 *레이블-기능 벡터* 쌍의 집합이어야 하며, 여기서 "기능 벡터"는 입력 지점을 나타내는 숫자의 벡터입니다. 따라서 반구조화되고 자유로운 형식의 주석이 많이 포함된 "위반(violation)" 열을 컴퓨터가 쉽게 이해할 수 있는 실수 배열로 변환해야 합니다.

자연 언어를 처리하기 위한 표준 기계 학습 접근 방법 중 하나는 고유한 각 단어에 "인덱스"를 할당한 다음 각 인덱스의 값이 해당 단어의 상대 빈도를 텍스트 문자열에 포함하도록 기계 학습 알고리즘에 벡터를 보내는 것입니다.

```
tokenizer = Tokenizer(inputCol="violations", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

model = pipeline.fit(labeledData)
```

1. 각 문자열에서 개별 단어를 가져오기 위해 각 위반 문자열을 "토큰화"합니다.
2. HashingTF을 사용하여 모델을 생성하는 로지스틱 회귀 알고리즘에 전달될 수 있는 기능 벡터로 각 토큰 집합을 변환합니다.
3. "파이프라인"을 사용하여 이 모든 단계를 차례로 수행합니다.

앞의 모델을 이용해서 뒤에서 예측하는 모델 만들기

모델에서 생성한 예측을 포함하는 새 데이터 프레임 **predictionsDf**를 만듭니다. 이 조각은 데이터 프레임을 기반으로 **Predictions**라는 임시 테이블도 만듭니다.

```
testData =  
sc.textFile('wasb:///HdiSamples/HdiSamples/FoodInspectionData/Food_Inspe  
ctions2.csv').map(csvParse) W.map(lambda l: (int(l[0]), l[1], l[12], l[13]))  
  
testDf = spark.createDataFrame(testData, schema).where("results = 'Fail'  
OR results = 'Pass' OR results = 'Pass w/ Conditions'")  
predictionsDf  
=model.transform(testDf)predictionsDf.registerTempTable('Predictions')  
predictionsDf.columns
```

```
# -----  
# THIS IS AN OUTPUT  
# -----  
  
['id',  
 'name',  
 'results',  
 'violations',  
 'words',  
 'features',  
 'rawPrediction',  
 'probability',  
 'prediction']
```

예측한 것들 중 하나를 살펴보는 명령어

```
predictionsDf.take(1)
```

model.transform() 메서드는 스키마가 같은 모든 새 데이터에 동일한 변환 방법을 적용하여 데이터 분류 방법에 대한 예측에 도달합니다. 몇 가지 간단한 통계를 수행하여 예측의 정확도를 알아볼 수 있습니다.

```
numSuccesses = predictionsDf.where("""(prediction = 0 AND results = 'Fail') OR  
(prediction = 1 AND (results = 'Pass' OR
```

```
results = 'Pass w/ Conditions'))""").count()
```


```
numInspections = predictionsDf.count()
```

```
print "There were", numInspections, "inspections and there were", numSuccesses, "successful  
predictions"
```


```
print "This is a", str((float(numSuccesses) / float(numInspections)) * 100) + "%", "success rate"
```

```
# -----  
# THIS IS AN OUTPUT  
# -----
```


```
There were 9315 inspections and there were 8087 successful predictions  
This is a 86.8169618894% success rate
```

PySpark  복사


```
%%sql -q -o true_positive  
SELECT count(*) AS cnt FROM Predictions WHERE prediction = 0 AND results = 'Fail'
```

PySpark  복사

```
%%sql -q -o false_positive  
SELECT count(*) AS cnt FROM Predictions WHERE prediction = 0 AND (results = 'Pass'
```

PySpark  복사

```
%%sql -q -o true_negative  
SELECT count(*) AS cnt FROM Predictions WHERE prediction = 1 AND results = 'Fail'
```

PySpark  복사

```
%%sql -q -o false_negative  
SELECT count(*) AS cnt FROM Predictions WHERE prediction = 1 AND (results = 'Pass'
```

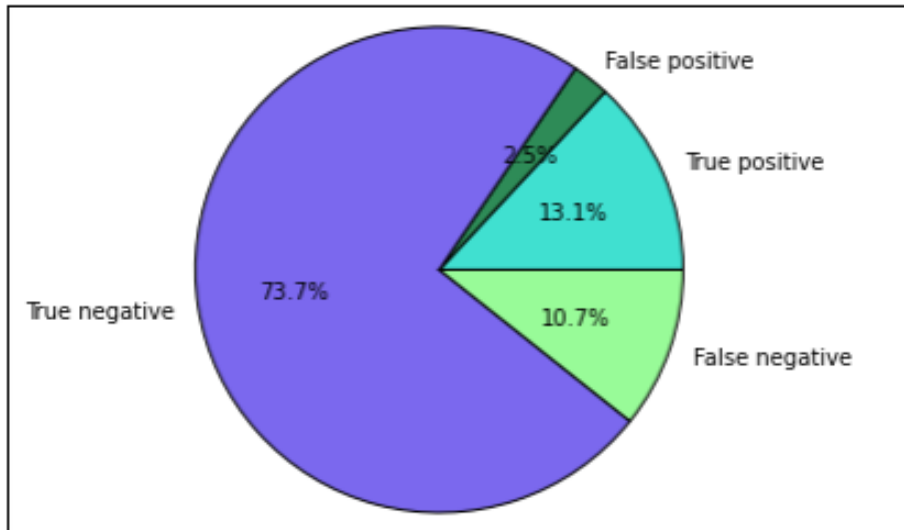
먼저 이전에 만든 **Predictions** 임시 테이블에서 여러 예측 및 결과를 추출합니다. 다음 쿼리는 출력을 *true_positive*, *false_positive*, *true_negative* 및 *false_negative*로 구분합니다. 아래 쿼리에서는 -q를 사용하여 시각화를 해제하고 %%local 매직에서 사용할 수 있는 데이터 프레임으로 출력을 저장(-o를 사용하여)합니다

```
%%local %matplotlib inline
import matplotlib.pyplot as plt
```

```
labels = ['True positive', 'False positive', 'True negative', 'False negative']
sizes = [true_positive['cnt'], false_positive['cnt'], false_negative['cnt'], true_negative['cnt']]
```

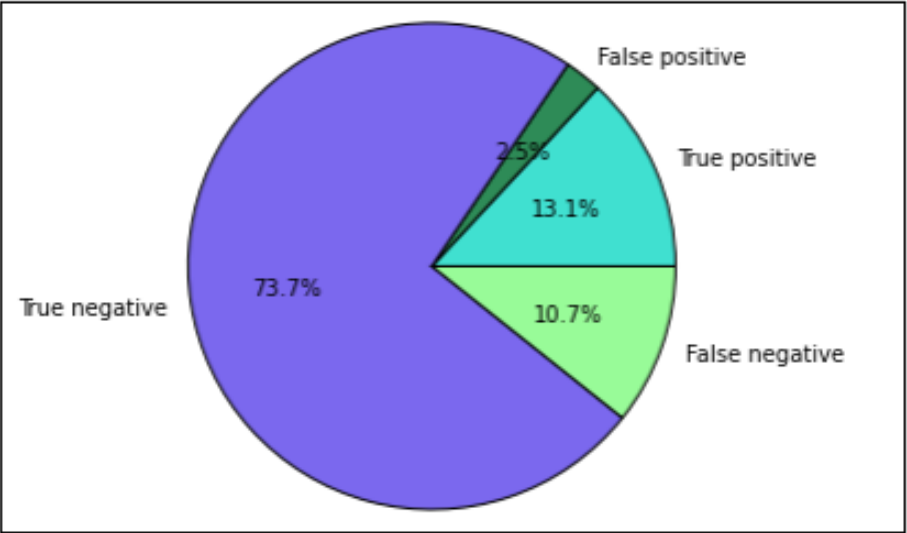
```
colors = ['turquoise', 'seagreen', 'mediumslateblue', 'palegreen', 'coral']
```

```
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors)
plt.axis('equal')
```



Positive = 음식 불합격
Negative = 음식 합격

True Negative 73.7% : 합격으로 예측 / 실제로 합격
 True Positive 13.1% : 음식 불합격으로 예측 / 실제로 불합격
 False Negative 10.7% : 음식 합격으로 예측 / 실제로 불합격
 False Positive 2.5% : 음식 불합격으로 예측 / 실제로 합격



Positive = 음식 불합격
 Negative = 음식 합격

		실제 결과(분류)	
		참	거짓
추론된 결과(분류)	참	TP (true positive)	FP (false positive)
	거짓	FN (false negative)	TN (true negative)