

# Training Models in Parallel using Pandas UDF in PySpark

```
%md
## Training Models in Parallel using Pandas UDF in PySpark
```

## Training Models in Parallel using Pandas UDF in PySpark

```
from sklearn.datasets import make_classification
import pandas as pd
import numpy as np
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from pyspark.sql.types import StructType, StructField, IntegerType, FloatType
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
from pyspark.sql.functions import pandas_udf, struct, PandasUDFType
```

### Create dummy data with 100 ids, each of them having 100 rows and 30 features

```
frames = []
for id in range(100):
    X, y = make_classification(n_samples = 100, n_features = 30, n_classes = 2, weights = [0.5,0.5], random_state = id)
    X_y = np.append(X, y.reshape(-1, 1), axis = 1)
    df = pd.DataFrame(X_y, columns = ['x' + str(i) for i in range(30)] + ['y'])
    df['id'] = id
    frames.append(df)

final_df = pd.concat(frames)
```

### Show sample data

```
z.show(final_df.head())
```

x0	x1	x2	x3	x4	x5	x6	x7
-2.7863740396742367	-0.6894491845499376	-0.6522935999350191	-1.8430695501566485	-1.2140774030941206	-0.4779740040404867	0.9318483741143037	0.698457149107336
1.0719956418304486	-0.13482245109435406	-0.5840935467949193	-2.437564359199356	0.8962484253561593	1.11492455949028	-1.5914478754580315	-0.361113134739866:
-0.3653805823167333	0.4046954556143003	0.19145087202391178	-0.45408036251560513	-1.962732009140752	0.08595197343438468	0.25280350541915453	-1.194986805268660:
0.30616506646092834	0.5726133530407902	-0.8612155533986415	1.0985816482311466	0.1416671648715303	-0.12706716283851693	-0.22157439820400068	0.7538656835623008
-0.4756834722241752	0.5311783665356953	-0.12824197402770202	0.21717963263828013	-0.6023318535828621	0.07821118109215218	1.5187593369635801	-1.481245962197298:

### 10K rows, 30 features, a response and an id column

```
final_df.shape
(10000, 32)
```

### We have 100 ids

```
final_df.id.nunique()
100
```

### Define schema, the features and response column

```
schema = StructType([
    StructField('id', IntegerType()),
    StructField('recall', FloatType()),
    StructField('precision', FloatType()),
    StructField('accuracy', FloatType()),
    StructField('auc', FloatType())
])

X_columns = final_df.drop(columns = ['id', 'y']).columns
y_columns = 'y'
```

### Create spark dataframe from the pandas dataframe

```
final_df_spark = spark.createDataFrame(final_df)
```

```
final_df_spark.count()
10000
```

```
%md
## Gradient Boosting Classifier
```

## Gradient Boosting Classifier

### Pandas UDF: a function that trains Gradient Boosting Classifier classifier using cross validation with random search and gets accuracy, recall, precision and area under the curve for each id

```
@pandas_udf(schema, PandasUDFType.GROUPED_MAP)

def model_results_per_id(df):
    id = int(df.id.unique()[0])
    X = df[X_columns]
    y = df[y_columns]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
    steps = [('scaler', StandardScaler()),
             ('gbm', GradientBoostingClassifier(random_state=0))]
    pipeline = Pipeline(steps)

    param_distributions = {'gbm__loss': ['deviance', 'exponential'],
```

## Training Models in Parallel using Pandas UDF in PySpark

```
        'gbm__learning_rate': np.arange(0.01, 0.5, 0.01),
        'gbm__n_estimators': range(20, 1000, 10),
        'gbm__max_depth': range(5, 100, 5),
        'gbm__max_features': ['sqrt', 'log2', None],
        'gbm__min_samples_split': [2, 5, 10]}

gbm_cv = RandomizedSearchCV(pipeline, param_distributions, cv = 5, n_iter = 100, n_jobs = -1, scoring = 'f1')
gbm_cv.fit(X_train, y_train)
y_pred = gbm_cv.predict(X_test)
accuracy = accuracy_score(y_test, y_pred).tolist()
precision = precision_score(y_test, y_pred).tolist()
recall = precision_score(y_test, y_pred).tolist()
y_pred_prob = gbm_cv.predict_proba(X_test)[:,-1]
auc = roc_auc_score(y_test, y_pred_prob).tolist()
model_results = pd.DataFrame([id, recall, precision, accuracy, auc], columns = ['id', 'recall', 'precision', 'accuracy', 'auc'])
return model_results

model_results_by_id = final_df.spark.groupBy('id').apply(model_results_per_id).toPandas()

###

#### As you can see from the time take above, it took about 8 minutes to train the models for all of the ids.
```

As you can see from the time take above, it took about 8 minutes to train the models for all of the ids.

### Display Sample results

```
model_results_by_id[['recall', 'precision', 'accuracy', 'auc']] = model_results_by_id[['recall', 'precision', 'accuracy', 'auc']].round(3)
model_results_by_id.sort_values(by = 'id').reset_index(drop = True).head()
```

	id	recall	precision	accuracy	auc
0	0	1.000	1.000	0.867	0.958
1	1	1.000	1.000	0.900	0.911
2	2	0.923	0.923	0.900	0.897
3	3	0.900	0.900	0.733	0.878
4	4	1.000	1.000	0.933	0.928

### Sklearn: how long does it take using vanilla Sklearn

```
frames = []
for id in range(100):
    df = final_df[final_df.id == id]
    X = df[X_columns]
    y = df[y_columns]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
    steps = [('scaler', StandardScaler()),
             ('gbm', GradientBoostingClassifier(random_state=0))]

    pipeline = Pipeline(steps)

    param_distributions = {'gbm__loss': ['deviance', 'exponential'],
                           'gbm__learning_rate': np.arange(0.01, 0.5, 0.01),
                           'gbm__n_estimators': range(20, 1000, 10),
                           'gbm__max_depth': range(5, 100, 5),
                           'gbm__max_features': ['sqrt', 'log2', None],
                           'gbm__min_samples_split': [2, 5, 10]}

    gbm_cv = RandomizedSearchCV(pipeline, param_distributions, cv = 5, n_iter = 100, n_jobs = -1, scoring = 'f1')
    gbm_cv.fit(X_train, y_train)

    gbm_cv.fit(X_train, y_train)
    y_pred = gbm_cv.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred).tolist()
    precision = precision_score(y_test, y_pred).tolist()
    recall = precision_score(y_test, y_pred).tolist()
    y_pred_prob = gbm_cv.predict_proba(X_test)[:,-1]
    auc = roc_auc_score(y_test, y_pred_prob)
    model_results = pd.DataFrame([id, recall, precision, accuracy, auc], columns = ['id', 'recall', 'precision', 'accuracy', 'auc'])
    frames.append(model_results)
all_model_results_sklearn = pd.concat(frames)

###

#### As you can see from the cell above, sklearn without pandas UDF took about 31 minutes to train all the models. About 4 times slower than Pandas UDF.
```

As you can see from the cell above, sklearn without pandas UDF took about 31 minutes to train all the models. About 4 times slower than Pandas UDF.

### Sample results

```
all_model_results_sklearn['id'] = all_model_results_sklearn['id'].astype(int)
all_model_results_sklearn[['recall', 'precision', 'accuracy', 'auc']] = all_model_results_sklearn[['recall', 'precision', 'accuracy', 'auc']].round(3)
all_model_results_sklearn.sort_values(by = 'id').head()
```

	id	recall	precision	accuracy	auc
0	0	1.000	1.000	0.933	0.944
0	1	1.000	1.000	0.900	0.929
0	2	0.786	0.786	0.800	0.915
0	3	0.923	0.923	0.800	0.910
0	4	1.000	1.000	0.933	0.955

### Pandas UDF: a function that trains Randomforest classifier using cross validation with random search and gets accuracy, recall, precision and area under the curve for each id

```
@pandas_udf(schema, PandasUDFType.GROUPED_MAP)
def model_results_per_id_rf(df):
    id = int(df.id.unique()[0])
    X = df[X_columns]
    y = df[y_columns]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
    steps = [('scaler', StandardScaler()),
             ('rf', RandomForestClassifier(random_state=0, n_jobs=-1))]

    pipeline = Pipeline(steps)

    param_distributions = {'rf__n_estimators': [100, 200, 500, 800],
                           'rf__criterion': ['entropy', 'gini'],
                           'rf__max_features': ['auto', 'sqrt', 'log2', None],
                           'rf__min_samples_split': [2, 5, 10],
                           'rf__min_samples_leaf': [1, 2, 4],
                           'rf__bootstrap': [True, False]}

    rf_cv = RandomizedSearchCV(pipeline, param_distributions, cv = 5, n_jobs = -1, n_iter=100, scoring = 'f1')
    rf_cv.fit(X_train, y_train)
    y_pred = rf_cv.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred).tolist()
    precision = precision_score(y_test, y_pred).tolist()
    recall = precision_score(y_test, y_pred).tolist()
    y_pred_prob = rf_cv.predict_proba(X_test)[:,-1]
    auc = roc_auc_score(y_test, y_pred_prob).tolist()
    model_results = pd.DataFrame([id, recall, precision, accuracy, auc], columns = ['id', 'recall', 'precision', 'accuracy', 'auc'])
    return model_results
```

## Training Models in Parallel using Pandas UDF in PySpark

```
model_results_by_id_spark_rf = final_df.spark_session.sql("""select * from model_results_by_id_rf toPandas()""")
```

```
%md
```

```
### Random Forest with Pandas UDF took about 13 minutes.
```

**Random Forest with Pandas UDF took about 13 minutes.**

### Sample results

```
model_results_by_id_spark_rf[['recall', 'precision', 'accuracy', 'auc']] = model_results_by_id_spark_rf[['recall', 'precision', 'accuracy', 'auc']].round(3)
model_results_by_id_spark_rf.sort_values(by = 'id').reset_index(drop = True).head()
```

	id	recall	precision	accuracy	auc
0	0	1.000	1.000	0.933	0.977
1	1	1.000	1.000	0.900	0.969
2	2	0.846	0.846	0.833	0.811
3	3	0.917	0.917	0.767	0.887
4	4	1.000	1.000	0.933	0.959

### Sklearn: how long does it take using vanilla Sklearn

```
frames = []
for id in range(100):
    df = final_df[final_df.id == id]
    X = df[X_columns]
    y = df[y_columns]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
    steps = [('scaler', StandardScaler()),
             ('rf', RandomForestClassifier(random_state=0, n_jobs=-1))]
    pipeline = Pipeline(steps)

    param_distributions = {'rf__n_estimators': [100, 200, 500, 800],
                           'rf__criterion': ['entropy', 'gini'],
                           'rf__max_features': ['auto', 'sqrt', 'log2', None],
                           'rf__min_samples_split': [2, 5, 10],
                           'rf__min_samples_leaf': [1, 2, 4],
                           'rf__bootstrap': [True, False]}

    rf_cv = RandomizedSearchCV(pipeline, param_distributions, cv = 5, n_jobs = -1, n_iter=100, scoring = 'f1')
    rf_cv.fit(X_train, y_train)
    y_pred = rf_cv.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred).tolist()
    precision = precision_score(y_test, y_pred).tolist()
    recall = precision_score(y_test, y_pred).tolist()
    y_pred_prob = rf_cv.predict_proba(X_test)[:,-1]
    auc = roc_auc_score(y_test, y_pred_prob).tolist()
    model_results = pd.DataFrame([id, recall, precision, accuracy, auc], columns = ['id', 'recall', 'precision', 'accuracy', 'auc'])
    frames.append(model_results)
all_model_results_sklearn_rf = pd.concat(frames)
```

```
%md
```

```
### Sklearn without Pandas UDF took 61 minutes (7.63 times slower)
```

**Sklearn without Pandas UDF took 61 minutes (7.63 times slower)**

```
all_model_results_sklearn_rf['id'] = all_model_results_sklearn_rf['id'].astype(int)
all_model_results_sklearn_rf[['recall', 'precision', 'accuracy', 'auc']] = all_model_results_sklearn_rf[['recall', 'precision', 'accuracy', 'auc']].round(3)
all_model_results_sklearn_rf.sort_values(by = 'id').head()
```

	id	recall	precision	accuracy	auc
0	0	1.000	1.000	0.933	0.963
0	1	1.000	1.000	0.900	0.900
0	2	0.786	0.786	0.800	0.915
0	3	0.929	0.929	0.833	0.921
0	4	0.917	0.917	0.900	0.955

### Main Points

```
%md
```

```
#### When you are training models per group or per ID, using Pandas UDF in PySpark can save you lots of time.
```

```
#### The amount of time you save will depend on how many distinct groups(IDs) you have and the power of your cluster. If you have more groups(IDs) and if you can use more nodes, you can save lots of time.
```

```
#### To get the similar results from Spark and Python, make sure to set random seed for train_test_split and RandomizedSearchCV.
```

```
#### Your data should not be very skewed to only very few of the grouping field.
```

**When you are training models per group or per ID, using Pandas UDF in PySpark can save you lots of time.**

**The amount of time you save will depend on how many distinct groups(IDs) you have and the power of your cluster. If you have more groups(IDs) and if you can use more nodes, you can save lots of time.**

**To get the similar results from Spark and Python, make sure to set random seed for train\_test\_split and RandomizedSearchCV.**

**Your data should not be very skewed to only very few of the grouping field.**

```
%md
```