In [1]:
```
import sys
import itertools
from math import sqrt
from operator import add
from os.path import join, isfile, dirname

from pyspark import SparkConf, SparkContext
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

In [ ]:
```
!pip install gcsfs
!pip install plotly
```

In [3]:
```
conf = SparkConf()
sc = SparkContext.getOrCreate()
DATA_PATH = "gs://zw2624-bucket/input/subsample_data_3.csv"
OUTPUT_PATH = "gs://zw2624-bucket/output/"
```

In [4]:
```
spark = SparkSession \
    .builder \
    .appName("example") \
    .getOrCreate()
df = spark.read.csv(DATA_PATH, header=True, mode="DROPMALFORMED")
```

In [5]:
```
sc.setCheckpointDir('checkpoint/')
```

In [6]:
```
from pyspark.sql.types import DoubleType, IntegerType, StringType
df = df.withColumn("userId", df["userId"].cast(IntegerType()))
df = df.withColumn("movieId", df["movieId"].cast(IntegerType()))
df = df.withColumn("rating", df["rating"].cast(DoubleType()))
```

In [7]:
```
(training, test) = df.randomSplit([0.8, 0.2])
als = ALS(userCol="userId", itemCol="movieId", ratingCol="rating", nonnegative = True, implicit
Prefs = False)
```

In [8]:
```
param_grid = ParamGridBuilder() \
            .addGrid(als.rank, [10, 50, 100]) \
            .addGrid(als.maxIter, [5, 10, 20]) \
            .addGrid(als.regParam, [0.01, 0.05, 0.1]) \
            .build()
```

In [9]:
```
evaluator = RegressionEvaluator(metricName="mae", labelCol="rating", predictionCol="prediction"
)
```

In [10]:
```
cv = CrossValidator(estimator=als,
                    estimatorParamMaps=param_grid,
                    evaluator=evaluator,
                    numFolds=5,
                    collectSubModels=True)
```

In [ ]:
```
model = cv.fit(training)
print('training finish')
best_model = model.bestModel
```

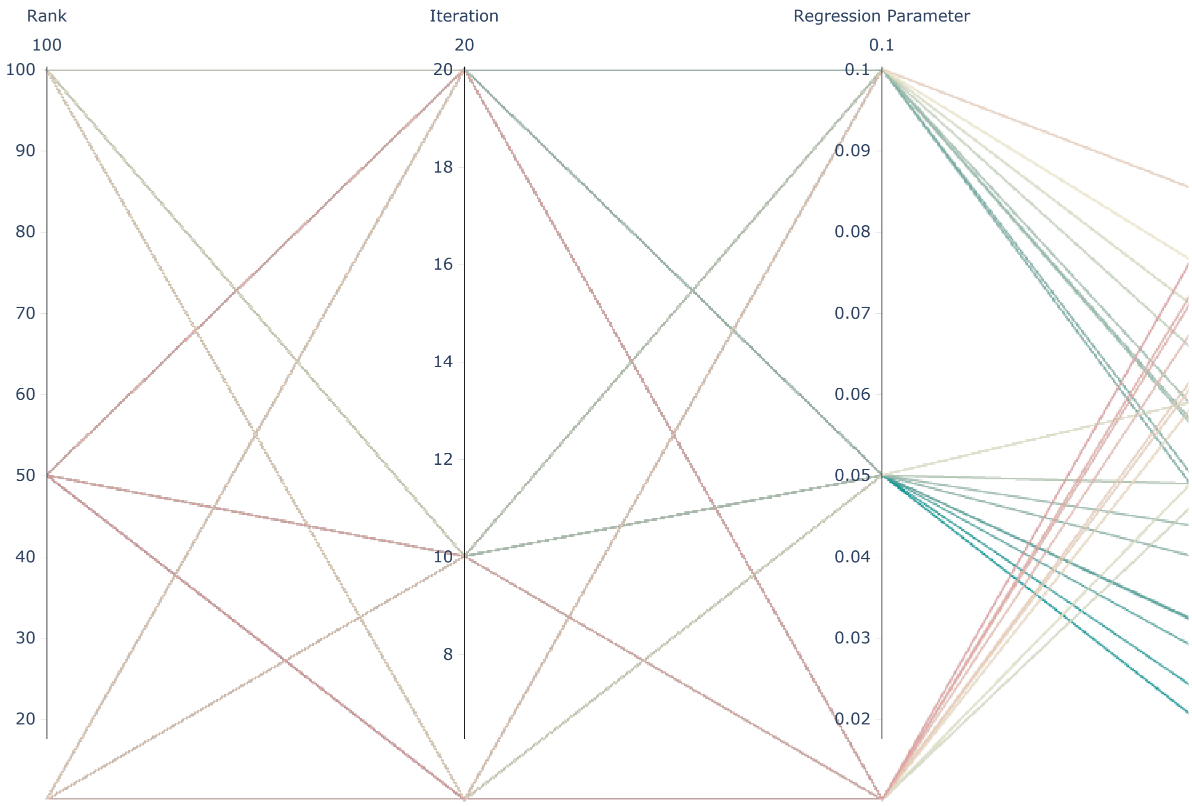
```
training finish
```

In [12]:

```
import pandas as pd
para_maps = model.getEstimatorParamMaps()
rank = []
maxIter = []
regParam = []
for i in range(27):
    values = list(para_maps[i].values())
    rank.append(values[0])
    maxIter.append(values[1])
    regParam.append(values[2])
result_df = pd.DataFrame({
    'rank': rank,
    'maxIter': maxIter,
    'regParam': regParam,
    'result':  model.avgMetrics
})
```

In [30]:

```
import plotly.express as px
import plotly.offline as pyo
import plotly.graph_objs as go
pyo.init_notebook_mode()
fig = px.parallel_coordinates(result_df, color="result", labels={"result": "Avg MAE",
                 "maxIter": "Iteration", "rank": "Rank",
                 "regParam": "Regression Parameter"},
                             color_continuous_scale=px.colors.diverging.Tealrose,
                             color_continuous_midpoint=0.66)
fig.show()
```



In [17]:

```
## Get user, item, catalog coverage.
def coverage(threshold1, threshold2, prediction):
    predictions = prediction.select("*").toPandas()
    pred = predictions.groupby('userId')
    df1= pred.apply(lambda x: x.sort_values(by=["prediction"],ascending=False))
    df2=df1.reset_index(drop=True)
    df3 = df2.groupby('userId').head(10)
    s1=df3[df3['rating'] > threshold1].groupby('userId')['rating'].count().reset_index()
    s2 = df3.pivot_table(index=['userId'],aggfunc='size').reset_index()
    s2.columns = ['userId','counts']
    df=pd.merge(s1, s2, on='userId')
    # #number of high true rating(larger than 4) devided by top N predictions
    df['rate']=df['rating']/df['counts']
    user_coverage=float(sum(df['rate']> threshold2))/df3['userId'].nunique()
    item=df3.groupby('movieId').apply(lambda x: x.sort_values(by=["prediction"],ascending=False
)).reset_index(drop=True)
    s=item[item['rating'] > threshold1].groupby('movieId')['rating'].count().reset_index()
    ss = item.pivot_table(index=['movieId'],aggfunc='size').reset_index()
    ss.columns = ['movieId','counts']
    dff=pd.merge(s, ss, on='movieId')
    dff['rate']=dff['rating']/dff['counts']
    item_coverage=float(sum(dff['rate']> threshold2))/df3['movieId'].nunique()
    catalog_coverage = float(df3['movieId'].nunique())/predictions['movieId'].nunique()
    return user_coverage, item_coverage, catalog_coverage

test_predictions = best_model.transform(test)
coverage(4, 0.5, test_predictions)
```

Out[17]:

```
(0.18066592898098782, 0.11102573953131002, 0.8682454969979987)
```

In [ ]:

```
dir(best_model)
```

In [31]:

```
evaluator.evaluate(test_predictions)
```

Out[31]:

```
0.6418457688710798
```