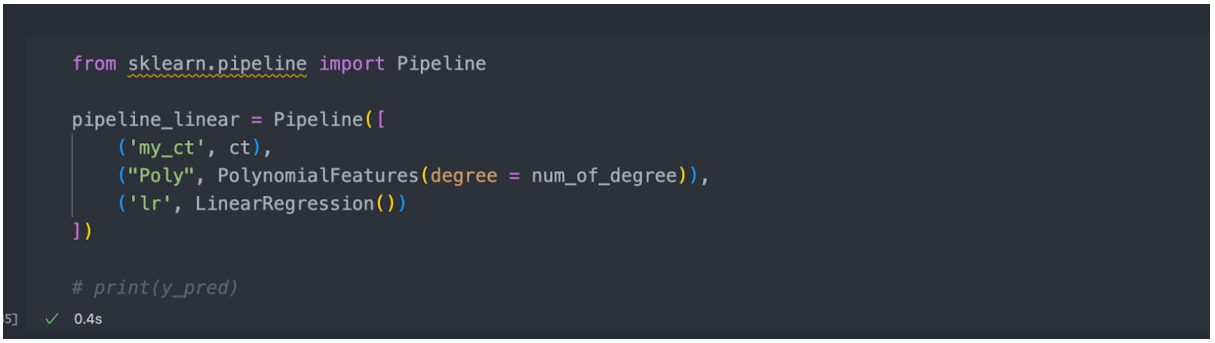
big data management exam report

jial@itu.dk

Pipelines

1. The pipeline is designed as shown following example:

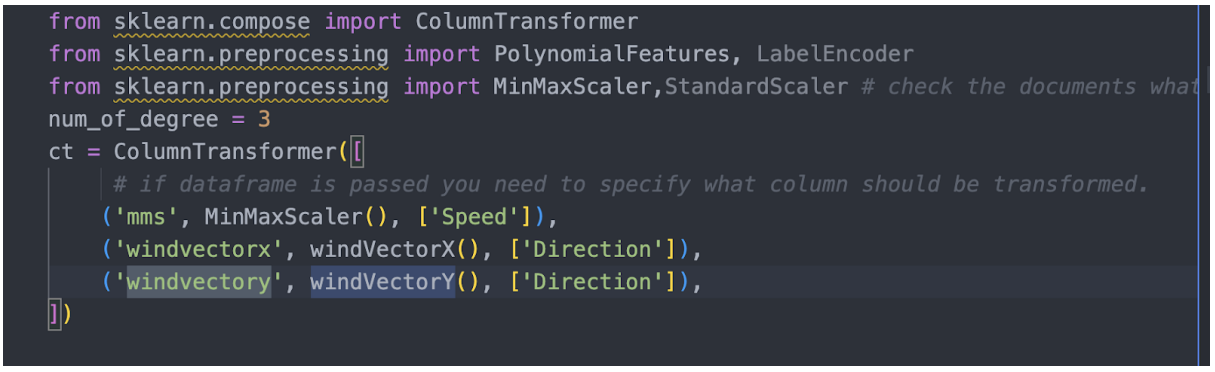


```
from sklearn.pipeline import Pipeline

pipeline_linear = Pipeline([
    ('my_ct', ct),
    ("Poly", PolynomialFeatures(degree = num_of_degree)),
    ('lr', LinearRegression())
])

# print(y_pred)
```

5] ✓ 0.4s

First, all the data is passed to a self-defined transformer named “ct”(which details will be shown latter) to make some preprocessing; Then it comes to polynomial features that add some more dimension to the whole dataset by using polynomial factors; Last, the pipeline will assign some different model to train on the processed dataset.

And for ct, the self-defined transformer has the following details as the code block shows:



```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import PolynomialFeatures, LabelEncoder
from sklearn.preprocessing import MinMaxScaler,StandardScaler # check the documents what
num_of_degree = 3
ct = ColumnTransformer([
    # if dataframe is passed you need to specify what column should be transformed.
    ('mms', MinMaxScaler(), ['Speed']),
    ('windvectorx', windVectorX(), ['Direction']),
    ('windvectory', windVectorY(), ['Direction']),
])
```

It contains three parts: The First one is a method that does min-max Normalization on the wind speed because the value of wind speed varies a very large range, therefore it is very necessary to make sure those data has to be the same scale before sending into our model. Then the following to parts mainly transform the wind direction from string to a wind vector which describe the direction in to dimension: degree to south/north, degree of east/west. I prefer this transformation more than using one-hot encoding because using on-hot encoding has too many redundant knowledge and two dimension is good enough.

The result: I choose two aspects for discussing: different parameter as well as different model.

For the first part, lets choose the step length when split the datasets using TimeSeriesSplit, from 5,10,to20. The result on KNN model is shown as follow:

```
    scores = []
    for i in range(len(x_train_list)):
        pipeline_KNN.fit(pd.DataFrame(x_train_list[i]), pd.DataFrame(y_train_list[i]))
        y_pred_KNN = pipeline_KNN.predict(pd.DataFrame(x_test_list[i]))
        print(f"Score: {round(pipeline_KNN.score(x_train_list[i], y_train_list[i]),2)}")
        scores.append (pipeline_KNN.score(x_train_list[i], y_train_list[i]))
    print("the mean of the score for split number: "+str(numSplit))
    print(sum(scores)/len(scores))
```

✓ 0.4s

```
Score: 0.73
Score: 0.72
the mean of the score for split number: 2
0.7275336287409642
```

```
the mean of the score for split number: 10
0.734185600923754
```

```
Score: 0.72
the mean of the score for split number: 20
0.7398924209385604
```

We could see the test score is rising slightly with split number increasing.

For the second part, we choose the different model: linear model, KNN and Random Forest, the result is as follows:

Linear:

```
    scores = []
    from sklearn.metrics import mean_squared_error as mse
    for i in range(len(x_train_list)):
        pipeline_linear.fit(pd.DataFrame(x_train_list[i]), pd.DataFrame(y_train_list[i]))
        y_pred_linear = pipeline_linear.predict(pd.DataFrame(x_test_list[i]))
        scores.append (pipeline_linear.score(x_train_list[i], y_train_list[i]))
    print("the mean of the score for split number: "+str(numSplit))
    print(sum(scores)/len(scores))
        # if i == 4:
        #     print(y_pred_linear)
```

80] ✓ 0.7s

```
the mean of the score for split number: 5
0.5567132150326446
```

KNN(with neighbors number 5):

```
Score: 0.73
Score: 0.72
the mean of the score for split number: 2
0.7275336287409642
```

Random Forest(max_depth=10, random_state=0):

81] ✓ 0.6s

```
the mean of the score for split number: 5
0.8450472335824426
```

We could see that KNN and Random Forest has more accuracy over linear model.

Scalable processing

Question1

1. First, I read the three CSV file, representing reviews, customers and business:

```
bs = spark.read.json("/datasets/yelp/business.json")
rs = spark.read.json("/datasets/yelp/review.json")
us = spark.read.json("/datasets/yelp/user.json")
```

2. Then, I use spark.filter to find the influencers that has written mare than 1000 reviews:

```
us_tmp = us.filter(us.review_count>1000).select("user_id")
us_tmp.show(10)
```

```
In [13]: us_tmp = us.filter(us.review_count>1000).select("user_id")
         us_tmp.show(10)

         +--------------------+
         |             user_id|
         +--------------------+
         |TEtzbpgA2BFBrC0y0...|
         |zzpgpo54-_P-4rzzB...|
         |eSlOI3GhroEtcbaD_...|
         |d7D4dYzF6THtOx9im...|
         |bURBDO2lgSrxnth2P...|
         |wQ0lKC5BMg3ZvCco8...|
         |U4INQZOPSUaj8hMjL...|
         |XkLK2iBsgqF6mWwDf...|
         |Ar3bat-NGasrXDiS7...|
         |fRJpK_b0rrjpBgRZj...|
         +--------------------+
         only showing top 10 rows
```

3. Then, I could combine the review content, business information as well as user information together:

```
q41 = rs.join(bs, on="business_id", how="inner").join(us_tmp,
on="user_id",how="inner").drop(bs.stars).select("name","business_id","rev
iew_id","review_count","longitude","latitude","city","state","categories"
,"text","stars").distinct()
```

4. In order to find those restaurant who has served more than 5 influencers, I defined another tool dataframe.

```
tool =
q41_distinct.groupBy("business_id").agg(count("*").alias("count"))
tool = tool.filter(tool[1]>5)
inf5_rev100 = q41_distinct.join(tool, on="business_id",
how="inner").drop(tool.business_id)
```

It has 3771 restaurants left:

```
tool.count()
```

```
FloatProgress(v
```

```
3771
```

The result is like:(It has 61518 records)

```
In [27]: %%pretty
         inf5_rev100.show(5)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

name	review_id	review_count	longitude	latitude	city	state	categories	text	stars	count
Delmonico Steakhouse	eA3hnRdUFMkA5UJyI...	1613	-115.16919	36.123183	Las Vegas	NV	Cajun/Creole, Sea...	My husband and I ...	4.0	49
Delmonico Steakhouse	jDpTiAR5GxTxEei0k...	1613	-115.16919	36.123183	Las Vegas	NV	Cajun/Creole, Sea...	If you want a per...	4.0	49
Delmonico Steakhouse	MjOgrAUgOrXooTqR2...	1613	-115.16919	36.123183	Las Vegas	NV	Cajun/Creole, Sea...	This place is per...	5.0	49
Delmonico Steakhouse	nm-nbTjqlwE0HJQSV...	1613	-115.16919	36.123183	Las Vegas	NV	Cajun/Creole, Sea...	See profile site ...	3.0	49
Delmonico Steakhouse	6ZeLaj4cuxJ6mp6RA...	1613	-115.16919	36.123183	Las Vegas	NV	Cajun/Creole, Sea...	I'll start with t...	2.0	49

```
only showing top 5 rows
```

```
In [13]: inf5_rev100.count()
```

```
61518
```

Qeustion2

First I choose 10 city that has most number of reviews to have a discussion:

```
In [41]: whole = spark.sql('''
                    select city, count(city) from inf5_rev100
                    group by city
                    order by count(city) DESC
                    ''')
         whole.createOrReplaceTempView("whole")
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
In [36]: whole.show(10)
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),…
```

```
+----------+-----------+
|      city|count(city)|
+----------+-----------+
| Las Vegas|      35678|
|   Phoenix|       4053|
|Scottsdale|       3581|
|   Toronto|       2524|
| Charlotte|       2282|
| Henderson|       1365|
|  Montréal|       1178|
|Pittsburgh|       1142|
|     Tempe|        986|
| Cleveland|        708|
+----------+-----------+
only showing top 10 rows
```

I choose the first five cities Las Vegas, Phoenix, Scottsdale, Toronto, and Charlotte as target. If you look at their positions, you'll find that Las Vegas,Phoenix and Scottsdal is in West of American, Toronto is in the north, while Scottsdale is in the East. According to live expensive level from https://www.bestplaces.net , the order is: Toronto > Phoenix and Scottsdale>Las Vegas > Charlotte.

Focusing on those cities, I decide to find the relationship between the percentage of "authentic" series words and the location of the areas. The SQL example is like this:

```
In [22]: Scottsdale = spark.sql('''
                    select * from inf5_rev100
                    where (city like '%Scottsdale%')
                    ''')
Scottsdale.createOrReplaceTempView("ScottsdaleView")
```

```
In [23]: %%sql
select count(*) from ScottsdaleView
```

Type: Table Pie

count(1)
3581

```
In [24]: %%sql
select count(*) from ScottsdaleView
where text LIKE "%legitimate%" OR text LIKE "%Legitimate%"
OR text LIKE "%LEGITIMATE%" OR text LIKE "%authentic%"
OR text LIKE "%Authen%" OR text LIKE "%AUTHEN%"
```

Type: Table Pie

count(1)
62

The result is shown as follows:

	Toronto	Scottsdale	Phoenix	Las Vegas	Charlotte
total	2524	3581	4053	35824	2282
authenic	67	62	61	511	38
percentage	26.55%	17.31%	15.05%	14.26%	16.65%

The table shows that the percentage of “authentic” language is the highest in Toronto, then is Scottsdale and Phoenix, then Las Vegas, and the lowest is in Charlotte. It seems that the North and Western area has more passion on using authentic language. I think the relationship not only lies in the location, but also because the local salary/purchasing level, so the richer a place is, the more possible that the people live in there will be picky on dishes, and more likely they will judge based on whether the food is more “authentic” or not.

ML lifecycle

Question1

I choose those four parameters to track:

```
32  # Import some of the sklearn modules you are likely to use.
33
34  modelType = sys.argv[1] if len(sys.argv) > 1 else 'KNN'
35  numSplits = int(sys.argv[2]) if len(sys.argv) > 2 else 5
36  numDegree = int(sys.argv[3]) if len(sys.argv) > 3 else 2
37  Dataset = int(sys.argv[4]) if len(sys.argv) > 4 else "dataBy09132022.csv"
38
```

1. modelType: What type of model to train. I choose it because we have to decide which model are using in the pipeline, which is very easy to change and reproduce.
2. numSplits: Step length of timeSeriesSplit() method on dividing dataset. The reason is the researcher of those prediction model will probably adjust the method of dividing a dataset, in which to get a more reasonable cross-validation result.
3. numDegree: choose the degree of the polynomial feature, making it easy to adjust the degree of polynomial feature that will be introduced into pipeline.
4. Dataset, choose the dataset of raw source data. I choose it because we need to compare which data file to be the source and make the experiments' reproducibility when comparing datasets on different time periods.

```
mlflow.log_param('number of splits', numSplits)
mlflow.log_param('degree of polynomial', numDegree)
mlflow.log_param('model of training', pipeline.steps[-1][0])
mlflow.log_param('Dataset', Dataset)
```

5. Besides, I choose different metrics to estimate the prediction results:MAE,MSE and R2-score.

```
# TODO: Currently the only metric is MAE. You should add more. What other metrics could you use? Why?
    metrics = [
        ("MAE", mean_absolute_error, []),
        ("MSE", mean_squared_error, []),
        ("R2", r2_score, [])
    ]
```

6. Last, a typical parameterized command will be like:
   python -u "/template.py" 'KNN',5,2,"dataBy09132022.csv"

   The result is shown like follows:

▾ Parameters

Name	Value
Dataset	dataBy09132022.csv
degree of polynomial	2
model of training	knn
number of splits	5

▾ Metrics

Name	Value
mean_MAE	6.142
mean_MSE	52.64
mean_R2	0.858

Question2

I will set up this experiment in such a edge-computing manner:

1. Hardware requirement: I will deploy a series of edge equipment(IOT endpoints, mobile CPU and so on) for every region. Let's call them nodes. All the nodes could communicate with the global datacenter, sending datasets to where the cloud server lies.

2. Software environment: I will use MLflow deploying to a cloud server and can be queried by many users. I will set up the tracking server, the backend store, and the artifact store reside on remote hosts, while those distributed edge nodes will preseve their ML model codes and communicated with remote host using MLflow clients.
3. Partitioning: dataset is collected in to a data lake, then hashing by key hash to store them into uniformly distributed, to avoid risk of skew and hot spots
4. Replication The cloud server will replicate its data together with several replicas in order to increase the safety, using strategy like raft protocol to keep replications up to date and fault-tolerant.
5. Scaling: the service will have some auto-scaling mechanism such as AWS Auto Scaling and S3 storage to monitor the service and automatically adjust capacity to maintain a consistent, predictable performance.
6. load balancing: It will use hadoop/spark computing cluster to map and deliver the task to computing nodes and the scheduler will make sure the pressure of working on multi-tasks is evenly distributed through the whole cluster.

Lecture material

1. Data lake and data warehouse differs in their data structures and functionalities. Data lake is a repository that stores raw data, which can be structured or unstructured, while Data warehouse stores highly curated data. Data lake is used to store the data after extraction, while Data warehouse is to provide data that are ready to be analyzed.

2. In ETL type data pipeline, data that collected from source, and then transforms it, and loads it into a destination data store, so the transforming step will using a separate transformation engine; However in the ELT pipeline, data loading is carried out combined with data transformation, so the transformation takes place in the target data store.
   When to use ETL example: The data size is relatively small and need complex transformations, or data that has a high level of privacy control and con not be directly transformed at target data storage, then choose ETL.
   When to use ELT example: The data size is very huge and needs relative simple transformations to analyze, also when the data is not privacy sensitive.

3. LSH: In some problem that need to calculate the distance between pairs of high-dimensional data points, this algorithm converts each data point as a sketch, instead of computing the distance between every pair, they just calculate the distance between sketches to find similarity between data points. Example: Classifying same trajectories based on driving records.

4. First the system also need a ELT module to extract data, transform and load to analysis module. I choose ELT because it could deal with huge size datasets more efficiently than ETL.
   Because we need to build a predictive maintenance system, it is necessary that we make sure that the real-time data must be used in order to predict possible crash/failures. So based on this point I design the system’s analytic part in a Lambda

architecture where speed-layer deal with stream data, and batch layer stores the raw data and performs batch process and form batch view. A serving layer is fed by both the batch and the speed layers.
Also the system will have some auto-scaling and load balancing mechanism such as AWS Auto Scaling and S3 storage in face of peak requests.

5. a. How distributed systems can guarantee safety/atomicy when they have to communicate with each other, probably on shared file read/writing?
   b. How could distributed replications maintain consistency, when update coming /some nodes commit/ the global leader dies?
   c. How to guarantee high performance while also make sure that once some partial nodes crashed, the service could still have correct functionality?
   d. How to schedule a distributed system so that they could have a consensus on one single logic clock, so that they may act as one deterministic order?

   bonus question:

   One of my favorite skill is XGBoost. It is a distributed big data analytic library. The idea of gradient-boosted decision tree make machine learning process could deploy on distributed nodes and compute in parallel, which make sure the efficiency as well as scalability on real world ML tasks.