Weather Forcasting Using Pyspark

Big Data Processing

Nguyen Phuoc Nguyen Phuc

Code github link: https://github.com/WinerDeCoder/Big-Data-Weather-ForeCasting.git

This project is inspired by: https://github.com/andrea-gasparini/big-data-weather-forecasting

I. Overview

This project’s goal is:

- Apply the power of distributed system (Spark) into big data processing
- Preprocessing the data, transformation, aggregate using Dataframe
- A ML model to predict weather condition based on some statistics

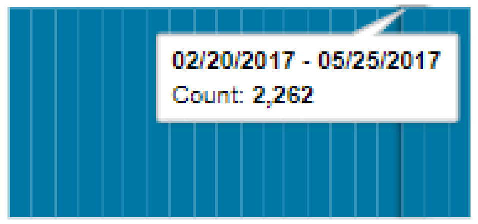
II. Data

A. Data overview

Data is taken from Kaggle contains hourly weather measurements data of 36 cities, collected from 2012 to 2017. This 5 years of data result in approximately 45.000 measurements (for each city) of temperature, humidity, air pressure and the like.

weather_description.csv (21.86 MB)

Detail Compact Column 10 of 37 columns

datetime	Vancouver	Portland	San Francisco	Seattle	Lo
 2012-10-01 2017-11-30	sky is clear 28% light rain 12% Other (26879) 59%	sky is clear 26% light rain 17% Other (25852) 57%	sky is clear 28% mist 18% Other (24525) 54%	sky is clear 28% light rain 16% Other (25290) 56%	sky i haze Othe
2012-10-01 12:00:00					
2012-10-01 13:00:00	mist	scattered clouds	light rain	sky is clear	mist
2012-10-01 14:00:00	broken clouds	scattered clouds	sky is clear	sky is clear	sky
2012-10-01 15:00:00	broken clouds	scattered clouds	sky is clear	sky is clear	sky
2012-10-01 16:00:00	broken clouds	scattered clouds	sky is clear	sky is clear	sky
2012-10-01 17:00:00	broken clouds	scattered clouds	sky is clear	sky is clear	sky
2012-10-01 18:00:00	broken clouds	scattered clouds	sky is clear	few clouds	sky

Data Explorer

Version 2 (74.69 MB)

- city_attributes.csv
- humidity.csv
- pressure.csv
- temperature.csv
- weather_description.csv
- wind_direction.csv
- wind_speed.csv

II. Data

A. Data overview

- The data set has 7 csv files
- The city_attributes.csv are only contain city and countries name, we won’t use this file as these are already in other files (however it can be used if you want to perform operation like join)
- Other 6 files, The weather_condition.csv are the weather condition we try to predict later. The rest are features we will use to predict

==> We will use only these features 'Country', 'Latitude', 'Longitude', 'humidity', 'pressure', 'temperature', 'wind_direction', 'wind_speed'

==> Our label is 'weather_condition'

B. Data preprocessing

We will process the data as following:

1. Load data on

```
weather_conditions_df = ks.read_csv('data/weather_description.csv')
humidity_df = ks.read_csv('data/humidity.csv')
pressure_df = ks.read_csv('data/pressure.csv')
temperature_df = ks.read_csv('data/temperature.csv')
city_attributes_df = ks.read_csv('data\city_attributes.csv')
wind_direction_df = ks.read_csv('data/wind_direction.csv')
wind_speed_df = ks.read_csv('data/wind_speed.csv')
```

1a. We drop some column of cities due to resource limitation of local computer (optional)

```
columns_to_drop = [ 'Charlotte', 'Miami',
       'Pittsburgh', 'Toronto', 'Philadelphia', 'New York', 'Montreal',
       'Boston', 'Beersheba', 'Tel Aviv District', 'Eilat', 'Haifa',
       'Nahariyya', 'Jerusalem']


weather_conditions_df = weather_conditions_df.drop(columns_to_drop, axis=1)
humidity_df = humidity_df.drop(columns_to_drop, axis=1)
pressure_df = pressure_df.drop(columns_to_drop, axis=1)
temperature_df = temperature_df.drop(columns_to_drop, axis=1)
wind_direction_df = wind_direction_df.drop(columns_to_drop, axis=1)
wind_speed_df = wind_speed_df.drop(columns_to_drop, axis=1)
```

```python
weather_conditions_melted = weather_conditions_df.melt(id_vars=['datetime'], var_name='city', value_name='weather_condition')
humidity_mel = humidity_df.melt(id_vars=['datetime'], var_name='city', value_name='humidity')
pressure_mel = pressure_df.melt(id_vars=['datetime'], var_name='city', value_name='pressure')
temperature_mel = temperature_df.melt(id_vars=['datetime'], var_name='city', value_name='temperature')
wind_direction_mel = wind_direction_df.melt(id_vars=['datetime'], var_name='city', value_name='wind_direction')
wind_speed_mel = wind_speed_df.melt(id_vars=['datetime'], var_name='city', value_name='wind_speed')
```

2. Join all tables together to form a consistance dataframe:

- In each table, we need to melt down so there no more column in each city - which is good for joining step because we will join based on “city”. After this, we should have all the table in form [’city’, ‘datetime’, ‘value’] (value is based on the feature)
- Then we join all table together on keys ‘city’ and ‘datetime’
- Then drop records that have null value

```python
join_kdf = weather_conditions_melted.join(city_attributes_df.set_index(['City']), on = 'city', how = 'left')  \
            .join(humidity_mel.set_index(['city', 'datetime']), on = ['city', 'datetime'], how = 'outer') \
            .join(pressure_mel.set_index(['city', 'datetime']), on = ['city', 'datetime'], how = 'outer')  \
            .join(temperature_mel.set_index(['city', 'datetime']), on = ['city', 'datetime'],how = 'outer')   \
            .join(wind_direction_mel.set_index(['city', 'datetime']), on = ['city', 'datetime'],how = 'outer')   \
            .join(wind_speed_mel.set_index(['city', 'datetime']), on = ['city', 'datetime'],how = 'outer')


join_kdf.columns
```

```python
not_null_weather_measurements_df = join_kdf.dropna()
```

3. There’re a lot of weather condition but similar to each other , we will replace all of it into 6 main weather: sunny, rainy, snowy, foggy, thunderstorm, cloudy

```
weather_mapping = {
    'sky is clear': 'sunny',
    'overcast clouds': 'cloudy',
    'light rain': 'rainy',
    'broken clouds': 'cloudy',
    'few clouds': 'cloudy',
    'haze': 'foggy',
    'very heavy rain': 'rainy',
    'thunderstorm with rain': 'thunderstorm',
    'smoke': 'foggy',
    'scattered clouds': 'cloudy',
    'proximity shower rain': 'rainy',
    'fog': 'foggy',
    'moderate rain': 'rainy',
    'proximity thunderstorm': 'thunderstorm',
    'light snow': 'snowy',
    'light shower snow': 'snowy',
    'snow': 'snowy',
    'sleet': 'snowy',
    'light shower sleet': 'snowy',
    'mist': 'foggy',
    'proximity thunderstorm': 'thunderstorm',
    'thunderstorm with heavy rain': 'thunderstorm'
}

not_null_weather_measurements_df['weather_condition'] =  \
not_null_weather_measurements_df['weather_condition'].replace(weather_mapping)
```

```
valid_values = ['thunderstorm', 'rainy', 'snowy', 'cloudy', 'foggy', 'sunny']
not_null_weather_measurements_df = not_null_weather_measurements_df[not_null_weather_measurements_df['weather_condition'].isin(valid_values)]
```

4. When counting weather condition, we should get like this. However, You can see the imbalance of label
=> We will downsample all other label down to the number of min label (6833)

```
weather_condition
rainy           121854
snowy             9931
sunny           511357
cloudy          419912
thunderstorm      6833
foggy           103404
Name: count, dtype: int64
```

```
spark_df = not_null_weather_measurements_df.to_spark()

# Specific number to downsample to
specific_number = value_counts.min()

# Downsample each group
def downsample_to_specific_number(df, col_name, count):
    return df.groupBy(col_name).applyInPandas(
        lambda pdf: pdf.sample(n=count, random_state=42) if len(pdf) > count else pdf,
        schema=df.schema
    )

downsampled_spark_df = downsample_to_specific_number(spark_df, 'weather_condition', specific_number)

# Convert back to Koalas DataFrame if needed
balanced_df = downsampled_spark_df.to_koalas()
```

III. Spark ML

Now we will use Pyspark for futher data analysis and Ml model

1. Create spark session
2. Load data from Koala_into Spark
3. Suffle the data
4. One-hot encode ‘weather_condition’ to get numeric label

```
import databricks.koalas as ks
from pyspark.sql import SparkSession
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql import SparkSession, functions as F

spark = SparkSession.builder.appName("KoalasAndSparkML").getOrCreate()

spark_df = balanced_df.to_spark()

spark_df = spark_df.orderBy(F.rand())

# Index the weather_condition column
indexer = StringIndexer(inputCol="weather_condition", outputCol="label")
indexed_df = indexer.fit(spark_df).transform(spark_df)
```

B. Data Analysis

Before fitting this data into ML model, we should perform some analysis on it first for better result and minimize cost

1. It's necessary to view the correlation of this dataframe
    a. We see the correlation of each features together
    b. Then the correlation of each feature to the label

=> We do this with Correlation Matrix on all dataframe

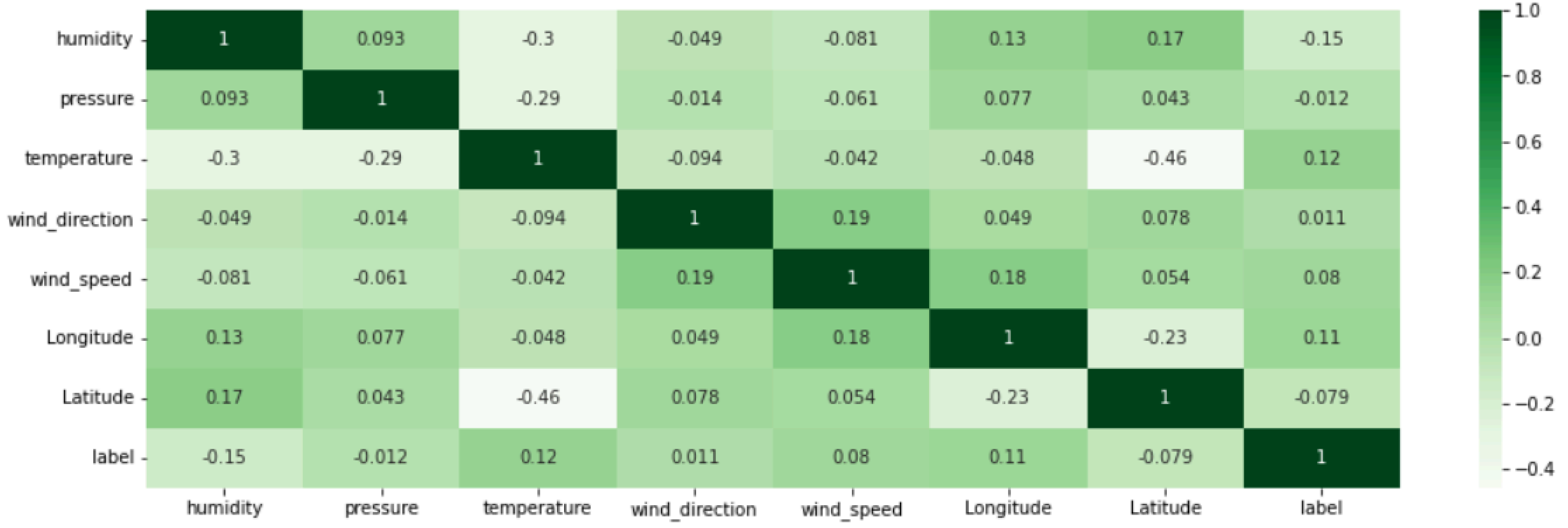
```
from pyspark.ml.stat import Correlation
from pyspark.ml.feature import VectorAssembler

# convert to vector column first
vector_col = "corr_features"
assemblerer = VectorAssembler(inputCols=['humidity', 'pressure','temperature','wind_direction','wind_speed','Longitude','Latitude','label'],\
    outputCol=vector_col)
df_vectorer = assemblerer.transform(indexed_df).select(vector_col)

# get correlation matrix
matrix = Correlation.corr(df_vectorer, vector_col)

matrixer = matrix.collect()[0]["pearson({})".format(vector_col)].values


import seaborn as sns
import matplotlib.pyplot as plt

x_labels = ['humidity', 'pressure','temperature','wind_direction','wind_speed','Longitude','Latitude','label']
y_labels = ['humidity', 'pressure','temperature','wind_direction','wind_speed','Longitude','Latitude','label']

plt.figure(figsize=(16,5))
sns.heatmap(matrixer.reshape(8,8),
            xticklabels=x_labels,
            yticklabels=y_labels,  cmap="Greens", annot=True)
```

2. Let see this correlation matrix:

- Basically, if the value is 0, that mean these 2 feature are completely independent. And if value in positive 0 < value <=1, that mean these 2 value are increasing dependent , and -1 <= value < 0 are decreasing dependent
- We want the correlation of features - label have relationship => Differ than 0. But we want correlation of feature - feature close to 0, because if 2 value is dependent, there is no sense to have both of them as feature, 1 is enough

=> As we can see

- For feature - label: only ‘humidity’, ‘temperature’, ‘wind_speed’, ‘Longitude’ and ‘Latitude’ seem have stronger relation ship to the label than other
- For feature - feature: Some really strong dependent are: Latitude - temperature, temperature - humidity, pressure - temperature, Latitude - Longtitude



3. Feature Selection + Dimensionally Reduction:

- We observe some feature contribute more in the resule ==> We only select these features.
- However, these features are have some high correlation pair => we need to dimensionally reduction the data. For better feature into the model, lower cost but with meaningful features.

I choose to use PCA as the reduction method, From 5 feature I will down to only 3.

** Actually i think LDA is a better method here, because the are caring about feature label => It make more sense to separate these class. However, pyspark doesn't implement it yet so PCA is just fine.

```python
from pyspark.ml.feature import VectorAssembler, PCA
from pyspark.ml import Pipeline

assembler = VectorAssembler(
    inputCols=['humidity', 'temperature', 'Longitude', 'wind_speed', 'Latitude'],
    outputCol='features'
)
feature_df = assembler.transform(indexed_df)

pca = PCA(k=3, inputCol='features', outputCol='pca_features')
pca_model = pca.fit(feature_df)
pca_df = pca_model.transform(feature_df)
```

III. ML model with Pyspark

For this task, to simplify, we will use Decision Tree as our Classification model (max_height = 5)
The idea of Decision Tree is :

- In each split, it will chose the feature split that maximize the information gain, try to make child node impurity. Then keep going into child node

The flow of using pyspark ML

1. Assembler defining “features” for ML model
2. PCA
3. Split the data into train/test with propotion 0.8/0.2
4. Put into the model
5. Testing result

```python
from pyspark.ml.feature import VectorAssembler, PCA
from pyspark.ml import Pipeline

assembler = VectorAssembler(
    inputCols=['humidity', 'temperature', 'Longitude', 'wind_speed', 'Latitude'],
    outputCol='features'
)
feature_df = assembler.transform(indexed_df)

pca = PCA(k=4, inputCol='features', outputCol='pca_features')
pca_model = pca.fit(feature_df)
pca_df = pca_model.transform(feature_df)
```

```python
train_dfer, test_dfer = feature_df.randomSplit([0.8, 0.2])
```

```python
# Train a decision tree classifier
dter = DecisionTreeClassifier(featuresCol='features', labelCol='label', maxDepth=5 )
dt_modeler = dt.fit(train_dfer)

predictionser = dt_modeler.transform(test_dfer)

# Evaluate the model
evaluatorer = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracyer = evaluator.evaluate(predictionser)
print(f"Test set accuracy = {accuracyer}")
```

III. Evaluate

Result on test dataset after feature selection + dimensionally reduction

```
Test set accuracy = 0.5053239255933291
```

Result on test dataset with all features

```
Test set accuracy = 0.5239704593816498
```

=> We get the result above 50% among 6 classes, which is not so bad on Decision Tree with max 5 height

=> Note that we should tuning the model futhur - Like Adjust max_height of the tree, the method of calculating Impurity . However, my local computer is not efficiently do that so I will skipp

IV. Deploy the model and use on real-time

It is important to save the model to future use:

- I save the model
- Then load if use later

Real-time weather forecasting

Now it’s time to do that in real world, we will collect data from real-time, use spark to process them quickly, then use the model to predict what is the weather currently

We will:

- Call API to get current weather data from many cities , we call every 5 minutes
- Preprocessing them to forming dataframe that similar to train/test set (using Pyspark ofcourse)
- Use model to predict weather
- We will also use the true label to test the model again

See the code in: **all_api_in_one**

We get accuracy of 0.52 in the first api request

```
accuracy = 0.5227443139215197
```

V. Summary and Future Work

So far, we have address the weather forcasting using ML and Big Data Tools

There're still a lot of things we can improve this:

- Another ML / DL model
- Model Tuning
- More data collecting
- Better data Analysis techniques (LDA, StandardScaler,...)
- Finetuning model overtime when new data comes

*My local computer doesn't