ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

BigData - Activity recommandation platform

A real-time service to find the best activity in you vicinity

---

Roman Shirochenko

Coppex Gilles

Constantin Victor

Vinh Mau Baptiste

Frédéric Bonnand

Drobnjak Nemanja

Maslov Ivan

Big Data Project,
Supervisor: Khasanova Renata
Professor: Koch Christoph

Contents

1 Introduction

Written by Gilles Coppex and Roman Shirochenko

1.1 Initial projects

When we started the project, we merged two different teams. The first one wanted to use temperature dataset to predict the best place for setting up windplants on the map and the second one proposed the prediction of the best place in Switzerland to go skiing. We experienced data retrieval problem very shortly: first the switzerland data proposed were not accessible even after several mails and discussion with the people working it that institute. Then about the first team idea, the TA and the Professor found this project not interesting enough. So we were forced to change our plans.

1.2 Final project idea

Then we came with another much more interesting idea: instead of working only on temperature or (exclusive or) precipitation (rain and snow) we could simply work on *both*. We then imagined a recommendation web-application that would be able to predict the best leisure activity for a given user at a given place and date based on the weather conditions. After searching the available for the touristic and leisure places datasets, we chose Openstreetmap dataset to parse a list of interesting points on the map. Then, as we will see, the MongoDB database will record this tuples and allow us to query them on user's demand.

We then coupled the weather prediction to filter out irrelevant activities: for example it is not interesting to spend your Sunday afternoon at a park near to your location if it is raining. For the weather prediction two datasets from National Climatic Data Center were used.

As we we will see in details, this filter is really important to improve our recommendations in regard to user's need.

1.3 Teamwork, workload

We started the project at 13 people and finished with only 7. We then listed the skills of every people and affected them to what they wanted to work on or on what they were good at. When people wanted to learn new skills we always tried to affect them in a pair with an already experimented co-worker so that they can acquire new knowledge without slowing down the evolution of the project.

On the next section, we present the summary of what every people in the group did.

1.4 Tasks distribution

Roman Shirochenko,
Overall estimated hours spent: 160h

- NOAA Storms dataset and Special Sensor Microwave/Imager (SSM/I), Satellite(NOAA website) datasets review
- The Global Historical Climatology Network(GHCN) dataset filtering and reforming using Spark framework
- Weather Interpolation using MongoDB database
- Datasets with places of interests review
- Nearest places of interests searching and MongoDB database creation

- HTTP Rest service based on Spray Scala framework
- Project leading, scheduling

Gilles Coppex,
Overall estimated hours spent: 155h

- Machine Learning (MlLib Spark)
- Dataset 1 temperature prediction (Decision Tree MlLib Spark)
- Dataset 1 best statistical model with extension to three days prediction
- Recommandation algorithm (extract Openstreetmap tags & filtering with prediction)
- Report structure, text collecting & writting
- Meeting preparation during the project's development
- Help for final project presentation slides (graphs, plots, etc.)
- Project leading and job scheduling

Victor Constantin,
Overall estimated hours spent: 125h

- dataset 2 weather
- prediction trying (MlLib Spark)
- classification to decrease error (BoostedTree / Barometer approach)
- help running the models on the cluster

Baptiste Vinh Mau,
Overall estimated hours spent: 95h

- Dataset selection and accurate filtering
- Statistical help, error reducing & general math advice
- HDFS filesystem dataset structure (Bash scripting)
- Models training on cluster (Scala)
- Scheduled dataset updating (every day update & predict)
- Connexion from the webserver to the cluster (SCDP)

Frédéric Bonnand,
Overall estimated hours spent: 60h

- AngularJS, Jquery map, HTML, CSS, JSON
- Final project presentation

Drobnjak Nemanja,
Overall estimated hours spent: 50h

- AngularJS, Jquery map, HTML, CSS, JSON

Maslov Ivan,

Overall estimated hours spent: 70h

- Dataset 2 output file filtering (remove mismeasurements, old stations, etc.)

2 Datasets

2.1 Dataset 1

Written by Roman Shirochenko

2.2 NOAA Global Historical Climatology Network (GHCN)

NOAA Global Historical Climatology Network (GHCN) is an integrated database of climate summaries from land surface stations across the globe that have been subjected to a common suite of quality assurance reviews. The data are obtained from more than 20 sources. Some data are more than 175 years old while others are less than an hour old.

2.2.1 Datasets selection

The data for 1970-2015 years was downloaded from the NOAA ftp server (http://www1.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/) to hdfs cluster filesystem. For the dataset downloading the bash shell script was written. The total size of the downloaded data is 43.2 GB. The measurement was performed daily.

2.2.2 Datasets cleaning

After data filtering, the data was reformed from the line to column view. The example of one measurement for the one station in line view:

```
USS0007M05S,20150431,TMAX,283,,S,T,
USS0007M05S,20150431,TMIN,-117,,,T,
USS0007M05S,20150431,TOBS,-50,,,T,
USS0007M05S,20150431,PRCP,0,,,T,
```

Output result scheme :

```
station_id, date, TMIN, TMAX, PRCP, TOBS:
USS0007M05S, 20150431, 283, -117, 0, -50
```

The last step was saving the dataset by separate stations folders. So for each station, as the result, were formed the folder with station ID name and file inside this folder that contains all historical data for the particular station.

All steps for the GHCN dataset filtering and reforming were performed with program written on Spark framework.

2.3 Dataset 2

Written by Ivan Maslov

2.3.1 NOAA Integrated Surface Database (ISD)

The Global Historical Climatology Network (GHCN) is an integrated database of climate summaries from land surface stations across the globe that have been subjected to a common suite of quality assurance reviews. The data are obtained from more than 20 sources. Some data are more than 175 years old while others are less than an hour old.

The Integrated Surface Database (ISD) consists of global hourly and synoptic observations compiled from numerous sources into a single common ASCII format and common data model.

ISD was developed as a joint activity within Asheville's Federal Climate Complex. NCDC, with U.S. Air Force and Navy partners, began the effort in 1998 with the assistance of external funding from several sources. ISD integrates data from over 100 original data sources, including numerous data formats that were key-entered from paper forms during the 1950s - 1970s time frame.

2.3.2 Downloading the dataset

The data for 1980-2015 years was downloaded from the NOAA ftp server to hdfs cluster filesystem. For the dataset downloading the java application was written. The total size of the downloaded data is 105.9 GB. The measurement was performed hourly. After downloading we have been transformd data to readable format using applied to data format tool which we change for our needs.

For the weather prediction algorithm the data was cleaned and reformed. First, we filtered only for US stations and then data was filtered data only by active stations. Station was considered as active if it was active for the last month. Also we find that not all stations measures the pressure and precipitation, so stations was filtered out by this criteria too. After data filtering, the data was reformed from the line to column view:

```
USAF  WBAN YR--MODAHRMN DIR SPD TEMP DEWP    SLP   ALT    STP MAX MIN PCP01 PCP06 PCP24 PCPXX SD
010580 ***** 201501010000 160   4   29   27  993,0 *****  946,8 *** *** *****  0,01 ***** ***** **
010580 ***** 201501010300 130   9   29   27  990,6 *****  944,5 *** *** ***** ***** ***** ***** **
010580 ***** 201501010600 150   4   34   30  988,1 *****  942,6 ***  29 ***** *****  0,04  0,03 **
010580 ***** 201501010900 250  13   38   32  987,2 *****  942,1 *** *** ***** ***** ***** ***** **
```

And output result with scheme :

```
(station_id, date, TIME, SLP, PRCP)
619970-99999 201501010600 1010.5 0.00
619970-99999 201501020600 1015.3 0.00
619970-99999 201501030600 1011.2 0.38
619970-99999 201501040600 1011.2 0.16
619970-99999 201501050600 1000.95 0.94
619970-99999 201501060600 990.7 1.51
619970-99999 201501070600 974.2 0.57
619970-99999 201501080600 1009.7 0.03
```

2.3.3 Folder splitting

The last step was saving the dataset by separate stations folders. So for each station, as the result, were formed the folder with station ID name and file inside this folder that contains all historical data for particular station.

All steps for the GHCN dataset filtering and reforming were performed with program written on Java.

3 MachineLearning

Written by Gilles Coppex, Victor Constantin and Baptiste Vinh Mau

3.1 Introduction

The goal of this chapter is to explain the path we followed to predict the weather for the next day in order to filter the recommended activities. Obviously we are interested in the temperature (expressed in degree Celsius), the rain level (expressed in millimeters) and the pressure (expressed in hectoPascal). But we will see that this is not so simple in practice.

We have decided to rely only on a statistical model instead of working with a physical model. In theory a physical model would have smaller prediction errors but is way more complex to establish. We prefered using the statistical power of the huge amount of measurements we had in our two dataset to run a Machine learning project.

3.2 Temperature

Considering first the temperature, their values that we use for training come from the first dataset. From that dataset we only use the values for the maximal temperature and the minimal temperature. In this text file we also had a column named "*observedTemperature*" that we decided not to take into account due to the fact we had no information about the measurement time and way it was created. We thus prefered to keep the minimum and maximum temperature of a day. This will allow us to easily project the temperature across the day by giving us a range.

In order to predict the values for the minimal and maximal temperature we separated them and run twice the same following approaches:

3.2.1 Second degree kernelized LinearRegression

In our first approach, we considered modeling the temperature by only looking at the current date. Since the dataset contained temperature values for every day for multiple years, we wanted to see if we could only use the date as an input to train a model that predicts the temperature for the following date.

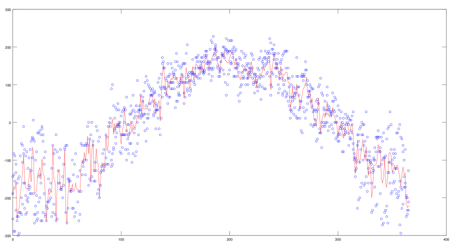
By plotting the temperature measurements with the date, we realized that the temperature follows a parabola throughout the year. We thus trained a regression model with a second degree kernel. Unfortunately this model did not give a good result at all.

3.2.2 Decision tree model

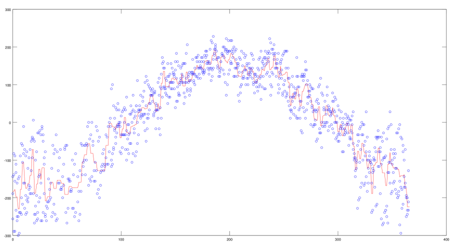
Our next approach was to use a Decision tree. One of the reasons that we chose to use a Decision tree is the fact that it can capture non-linearity, which is very useful in our case.

Due to the fact that a Decision tree partitions the feature space (in our case : days in the year), the number of bins / leafs will determine how many consecutive days will fall in the same bin. For example by choosing the number of bins to be 52 we will get 7 days in each bin. This allows us to predict over a frame of 7 days, but it also means that the prediction will be the same for these days.

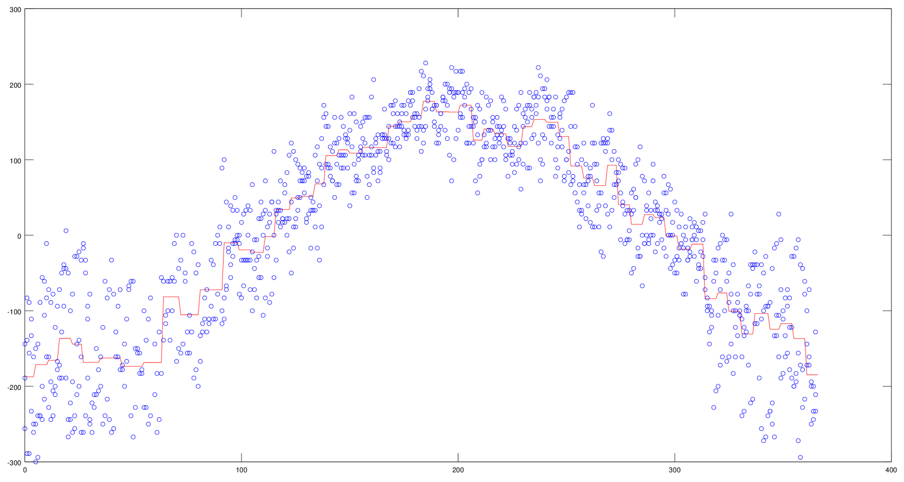
We empirically tested three different number of beans of our Decision tree in order to find the optimal amount of bins that give the best prediction. In our case the best number of bins was 50.



(a) DecitionTree with 366 bins



(b) DecitionTree with 180 bins



(c) DecitionTree with 60 bins

Figure 1: DecitionTree bin amount selection

On these graphs we see that 60 bins are suffisent to predict with a pretty good accuracy. However we can compute several Trees with different bin size and then plot the error:

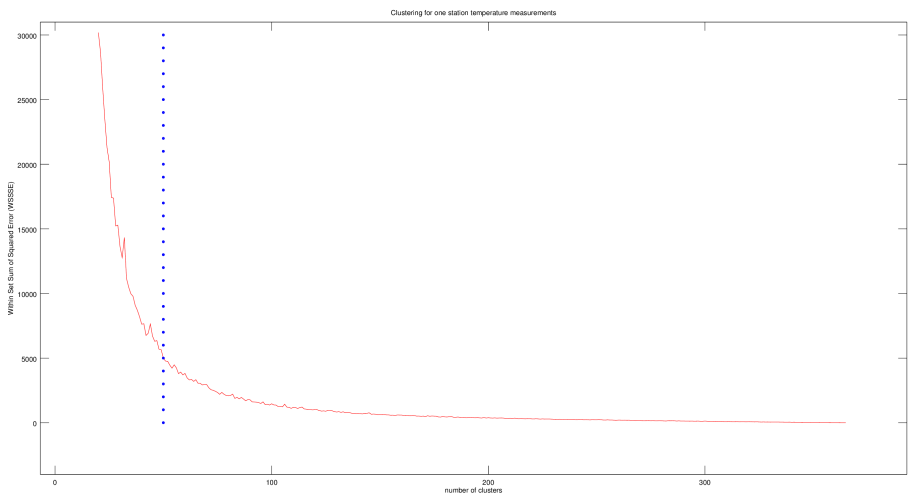


Figure 2: Error vs the number of bins in our DecisionTree model

On Figure 3, the error decreasing rate is almost zero when the number of bins goes over 60.

3.2.3 Temperature prediction taking previous days into account

However this approach has an important flaw: with only this model, every $\frac{366}{60} \approx 6$ days are mapped to the same prediction which is useless in practice. This section explains how we can address this issue.

We now want to analyze if the temperature can be predicted by looking at the previous measurements. The idea behind this is that the temperature should have a high inertia and that the temperature of today may not differ a lot for the one of tomorrow.

However the problem is that we can not predict rapid changes in the temperature due to some extreme conditions (storms, huricanes, etc.), but for cases where the weather does not change, it should work well.

The next big question is how many days do we need to take into account? In order to solve this question, we used a moving window of size n to get the n previous temperature values and use them as features to predict the temperature for day $n+1$.

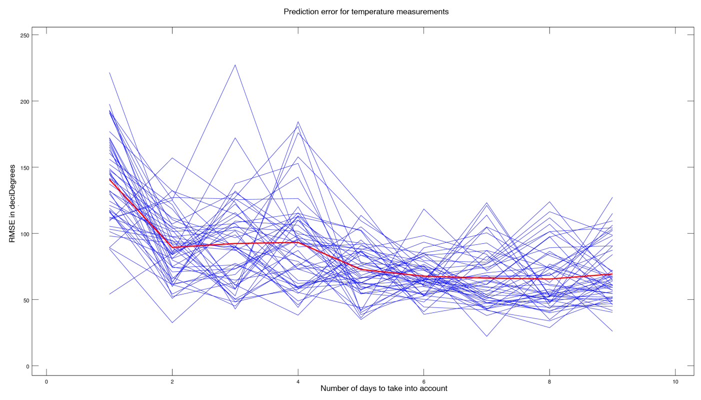


Figure 3: DecisionTree model cross validation on the number of days

We tried different values for n and used cross-validation over multiple splits to determine the performance for each value of n. The final value we empirically selected is 7. That is why use the seven previous days to determine the eighth one.

3.2.4 Prediction extension to 3 days

In this section, we explain how we can extend this model to be able to predict not just the temperature for the next day but also for the three following days. We tried two different methods and here are our results. The very first naive idea is to predict at all three days using a *moving window*: We predict the 8^{th} day base on the seven previous days and then consider this output as a new input for the prediction for the 9^{th} day temperature. We do the same for the 10^{th} day, every time moving the *7 day window*.

However this naive method have a statistical error cost: The prediction error will propagate at each step and thus the 10^{th} day prediction might be very far from the real value. Running on an arbitrary measured dataset, we obtain the following RMSE values :

Day	**RMSE**
Next day	5.49
Second Next day	6.89
Third Next day	8.05

We then try to improve these RMSE. We trained three different and distrincts Decision Tree models :

1. The first one takes 7 previous day temperature measurements and predicts the 8^{th} day.
2. The second one takes 7 previous day temperature measurements and predicts the 9^{th} day.
3. The thir one takes 7 previous day temperature measurements and predicts the 10^{th} day.

Obviousy, with this method, there is a gap between the predicted day and the days from which the measurements come from. This means that if some extreme event happens between them, the model does not know about it and can not capture it. Here is the RMSE for this statistical setup:

Day	**RMSE**
Next day	5.49
Second Next day	8.22
Third Next day	7.21

Despite the gap, the results of this second approach improved a bit for the third predicted day in regards to the first one (but worst performance for second predicted day). The trade-off of this model is that for each temperature value we have to predict we must train 3 models. This cost is prohibitive and the improvement is not that great to neglect this overhead. We thus prefer using the first approach (knowing that real life weather forecast is using analogue techniques).

Here is a typical final model plot:

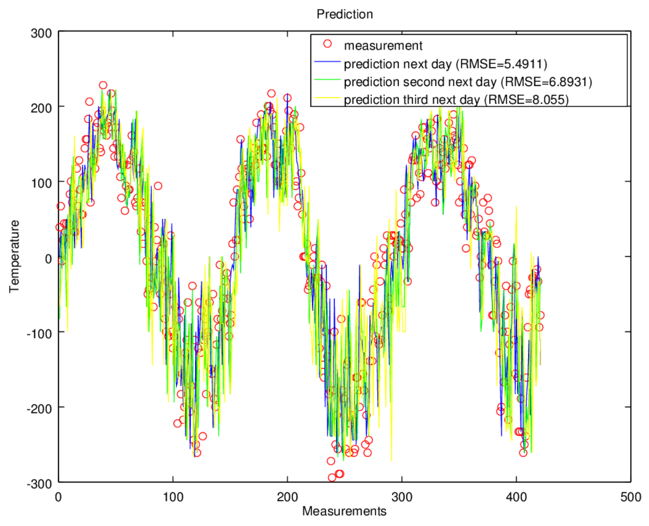


Figure 4: Temperature prediction for the 8th, 9th and 10th day

3.3 Rain measurement

The rain measurement is captured in the two dataset we selected. On dataset 2 however the amount of empty lines is a little more significant than in dataset 1. We reused the previous concept of the 7-day sliding window but the result were too poor. So we decided to do a cross-validation on the number of day to take into account and we detected that the best window size is one day only. The meaning behind this is that the best time model you can use to predict the rain level is to look at the previous day.

However the RMSE was still too high and then we realized this approach will always fail as the rain does not depend on the previous days: after two day of dry weather the third day could be rainy. So we changed our mind as we will see in the next chapter.

Another theoretical issue we can expect is the error handling: what can we do with a prediction measurement of, let's say, 5 mm of rain and an RMSE of 8 mm? We cannot just naively say it is raining as the error is too high and the rain level never goes beyond 0.

In the next section, we solved these two problem by a nice trick, our grandparents would be proud of.

3.4 Pressure prediction

3.4.1 First approach: consider the moving-average model

In our database the pressure is in hPa and is ranging from 960 to 1070. Our initial idea was to predict the pressure for the day and use it to predict the weather afterward.

Obviously we first used the same moving-window model as for the temperature, meaning that we predicted the pressure using the seven previous measurements. This idea is motivated by the fact that the pressure seems to be related between days, but practice showed us the opposite: this method gave an error (RMSE) of 40hPa, which represents around 0.4% of magnitude error.

Even if this value can be considered small relative to the real value, taking into account, the fact that only values between 960 and 1070 occur in real life, then the error represents 80% of the size of this smaller interval! This is a huge value and it means that whatever the predicted pressure value is, we are unable to correctly predict the pressure.

We changed the approach: instead of predict then classify, we decided to classify and then predict *the class* for the next day.

3.4.2 Second approach: add more input feature to the model

We can be more clever than just predict the next day pressure based on the previous day measurement: As the dataset 2 also contained the measurements of the precipitation, we decided to add them as input feature when training and predicting. By adding them, it could allow the model to at least improve its predictions with regard to bad weather. With this new model the RMSE did indeed go down and had a value 11, wich is not so bad.

But while plotting the predicted value vs. the real measurement (see Figure 5), we saw that the model tends to act as a local mean and can not keep up with the constant jiggling of the measured values and that in practice the result after the classification would be almost always the same.

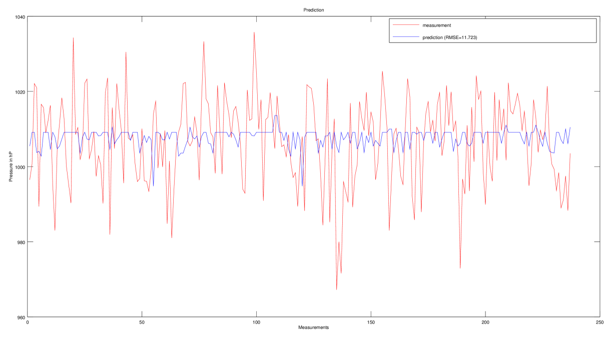


Figure 5: BoostedTrees model to predict next day pressure based on Precipitation and Pressure of one previous day

3.4.3 Third approach: our grandparents could have told us

Due to the previous section's problem, we changed our strategy.We thought about a barometer in order to classify the weather depending on the predicted pressure. A quick google image search and then we deduced the range empirically following the labels our ancestors defined.

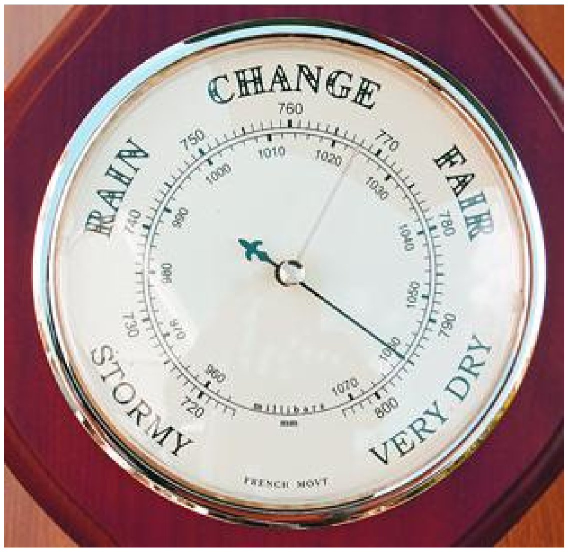


Figure 6: Classic barometer you can find on old building (image taken from the following website: http://italophiles.com/london_italians2.htm)

With this method we get the following classification :

Pressure	**Weather**
Smaller than 980	Storm
Between 980 and 1000	Rain
Between 1000 and 1025	Change
Between 1025 and 1050	Fair
Greater than 1050	Very Dry

Instead of predicting the pressure and then classifying the weather based on it, we would first classify our measurements and use those values to predict the next day class. We kept the same model as before, but this time, instead of feeding it pressure values we used the classes. Also this time it would return the class instead of the pressure. In the mean time we switched from a BoostedTree model to a RandomForrest one.

By applying cross-validation and testing for different range of previous day we found that by only using the weather class and the precipitation level of just the previous day the model would perform well. Indeed the model would predict the correct class 75% of the time and it would reach 97% if we also consider the times it predicts a neighboring class (fair vs. very dry). There can be a border effect when we predict either *changing* or *fair weather*. In these cases the 1 class error really has an impact, as our selection might be wrong. Nevertheless, we can still assume that 3 times out of 4 we will still exactly predict the right category and with an error of more than 1 class with 3% of probability, which is more than sufficient for our application.

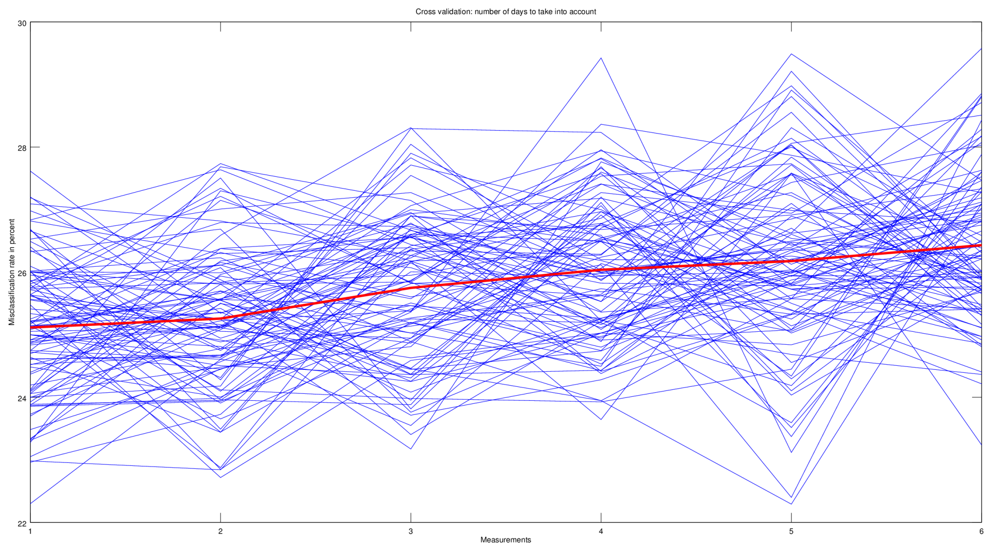


Figure 7: RandomForrest model cross validation on the number of days

3.5 Machine learning conclusion

In a sense these model perform very well as most of the times an neighboring class will have the same impact as the real class. As it will be explained better and more thoroughly in the recommendation system part of this report, the predicted temperatures and weather classes will be used to recommend activities to the user. The main impact of weather prediction is to choose between indoors and outdoors activities. A stormy, rainy or changing weather will select indoors activities and fair and very dry weather will select outdoors activities. Therefore by predicting stormy or raining we will still choose indoor activities and by predicting very dry we will choose outdoor activities.

4 Task Parallelization

4.1 Scaling the Machine Learning

After determined which ML model to use to predict each of the parameters, we had had to implements Spark-job to handle the training of all these models on all stations (around 4k).

One of the last problem that rose concerned the parallelization of the predictions algorithm. We ended by having two servers on which our project run. The course cluster contained all the classes and script related to the ML-part (as we need Spark to run the algorithm on our large amount of data) and an Azure server handled the rest. The problem was that both servers could only communicate via SSH sessions. So it was decided to reduce at most as possible the part that were handled by the cluster. For example, we could not decide to open a ssh connection with the cluster each time we needed to provide a prediction to an user.

So, a Spark-script has to generate the predictions every day at a fixed hour for all the station, and stored on a separated folder on the cluster. Then, the Azure server will download all this folder by using an SSH command. It is now possessing all the prediction he will need for the day, thus slightly reducing the number of needed SSH connections.

4.2 Daily Script Execution

One of the constraint that we had to take in account while designed our system is that some of task need to be run on a daily basis. On the final design, we had to execute each day a CRON script on the Azure server that will run the following task on the cluster:

1. Download the updated files of the both datasets.
2. Extract the data of the current day on the both datasets and split them by stations
3. Generate the predictions the day for each station
4. Download theses new predictions on the Azure Server...

This task shall not take more than a few hours, if it take more than this delay, we may not be able to deliver fresh predictions to the user at the start of each day. As discussed earlier, is it better to generate the prediction for all the stations at these step, as doing it each time as we had to retrieve a prediction for an user can cause delay up to a dozen of seconds.

5 Weather Interpolation

Written by Roman Shirochenko

5.1 Weather interpolation method

As we were worked with discrete stations, for the weather determination the interpolation was used. For the weather interpolation the Inverse Distance Weighting(IDW) method was chosen.IDW is an advanced nearest neighbor approach that allows including more observations than only the nearest points. The value at a certain grid cell is obtained from a linear combination of the surrounding locations.

The weight of each observation is determined by the distance, the distance function is non-linear. IDW is an exact interpolator [1].

By Inverse Distance Weighting(IDW) interpolation method the final value estimates with formula: $t = K * \sum_{i=1}^{i} t_i * d_i$ where:

- t_i – measurement of one station,
- $w_i = 1/d_i$ – weight(inverse euclidean distance from user location) of the station
- $K = 1/\sum_{i=1}^{i} d_i$ – normalization coefficient

5.2 Implemantion

For searching the nearest stations from the user location the NoSQL database MongoDB were used. The main reason for using MongoDB system was build-in support work with geospatial indexes. The two databases collections(one for GHCN and one for ISD dataset) with station coordinates and elevation were built and then a geospatial index created. This index was used for the search the nearest n stations and then for the temperature measurement. Also elevation of the stations was considered.

From the GHCN dataset we predict two values of temperature:maximum and minimum and then interpolation were used. For the ISD dataset we predict the classified weather conditions based on barometer classes and then the result were interpolated.

As we had the output from the model Tmin and Tmax for one day from dataset GHCN. We decide to use simple interpolation model of dependence Tmin and Tmax from the day time. The model represents as a graph on the figure 8.

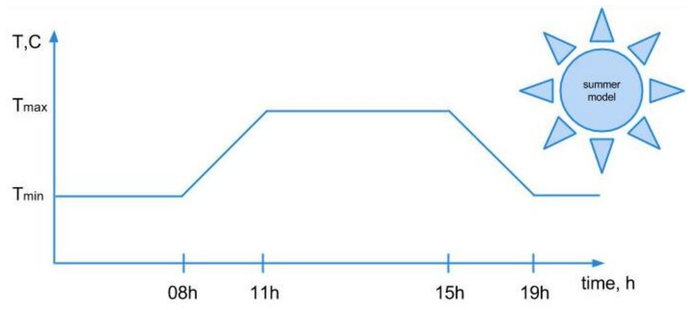


Figure 8: Temperature minimum and maximum interpolation model

6 Points of interests

Written by Roman Shirochenko and Baptiste Vinh Mau

We checked the different dataset for the places. Initial idea were to use foursquare data, but the dataset from arxiv.org did not contain any particular names of places. We did not want to use API service for data retrieval, so checked the Tripadvisor dataset for arxiv.org website. But it did not contais needed information also. Then we checked wikitravel.org and Openstreemap dataset.

6.1 Interest points datasets selection

As Openstreetmap dataset was provide more available places and this dataset actively refreshing by Openstreetmap community we decide to use this dataset. The dataset for the USA area was downloaded from http://download.geofabrik.de/. The URL for the list of the available features: http://wiki.openstreetmap.org/wiki/Map_Features. The format of the downloaded dataset .osm.pdf. Dataset was saved on hdfs cluster. The total dataset size for the USA places only is 8.3 GB.

From this original dataset, we started by trying to filtering most of the useless data. On OpenstreeMap, each places is characterized by some characteristic fields (ie sport, tourism...) that contains a list of tag that may (sport_centre) or may not (basketball) give us information about the fact that the place is inside a building. This will be useful for a latter part of the project. So we run the tools provided by OpenStreetMap to keep only a given set of characteristic. We also ran a Spark-Job to keep only the information on the places that were useful for our project (name, website...) and we merge all the tags of a different characteristics on a unique tag list, as the organization of the characteristic on OpenstreetMap doesn't fit our goals (for example the tag in amenity can be related to almost anything). We end up with a places dataset of a few dozen of MB.

6.2 Search for the nearest places

For searching the nearest places from the user location the NoSQL database MongoDB were used. We used the same approach as with weather stations, first we wrote the program on Scala that imports the places on interests to the MongoDB. But before the data was cleaned, because we have many points without names. After the geospatial index were created for the nearest places in the determined radius search performing.

7 Recommandation System

The recommendation system is responsible for retrieving activities based on two criteria:

- the activity should be in the vicinity of the user,
- the proposed interesting points should be filtered according to the weather conditions.

To do this task, this module can rely on the weather predictor and on the MongoDB that contains the geolocalized activities.

7.1 Inside activities

Our algorithm was two fold: first we determined whether the interesting point was in a building (namely we used some amenity tag to determine if the spot is inside). If this condition holds, we simply output the interesting point to the user (as an inside activity is almost never influenced by weather conditions).

7.2 Outside activities

Then if the activities near the user are outside, we need to query the weather recommendation system to get :

- the predicted temperature (t_{min} and t_{max} that has been interpolated to the current temperature, at time of query),
- the weather predicted class (from STORMY to VERY_DRY).

We then classified activities as the following:

- winter sports to lower temperatures (below 5 degree celsius)
- normal activities to normal temperatures (from 10 to 25 degree)
- extreme or water activities to highest temperatures (from 25 degree and above)

Then we took the weather predicted class into account to filter out non-suitable activities:

- if the weather will be stormy: we let no outside activities be recommended
- if it is raining: only outside water activities can be proposed to the user
- else we filter nothing out.

8 Web application

Written by Roman Shirochenko

We developed web-application where user can chose location where he wants to go or his current location. After this on the map will be displayed the recommended places for visiting considering the feature weather and distance from the user location. In this chapter will be described the back-end and front-end part of the web-application.

8.1 Web server technologies

For the points of interests data retrieval, the HTTP REST service were developed based on the Spray – Scala framework for building RESTful web-services on top of Akka were user [Spay framework officil website. URL:http://spray.io/].

The basic interface that was implemented contains one command:

GET /location/lat/long

The lat and long its coordinates of the user location that sends by the GET request from the web-service.

The response for this request contains the points of interests near the user location with considering of weather prediction facts and recommendation based on this weather prediction.

The work flow of the web-server can be described as follows:

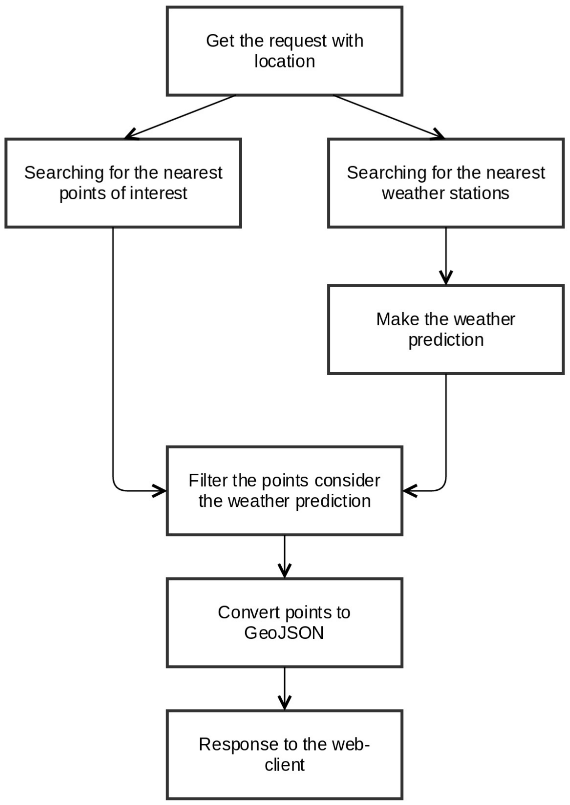

Get the request with location
Searching for the nearest points of interest
Searching for the nearest weather stations
Make the weather prediction
Filter the points consider the weather prediction
Convert points to GeoJSON
Response to the web-client

Figure 9: Web-application workflow diagram

8.2 User interface

For searching the nearest places from the user location the NoSQL database MongoDB were used. We used the same approach as with weather stations, first we wrote the program on Scala that imports the places on interests to the MongoDB. But before the data was cleaned, because we have many points without names. After the geospatial index were created for the nearest places in the determined radius search performing (see figure 10).

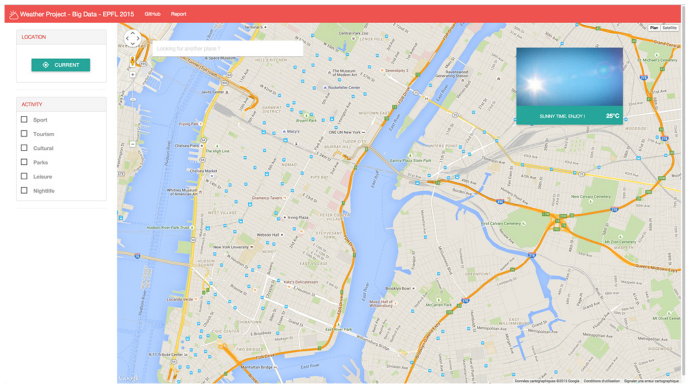


Figure 10: Web-application user interface

8.2.1 The web UI design

Written by Frederic Bonnand

We decided to display the interactive map throw a web browser. Consequently, the technologies used are HTML5, CSS3 et Javascript. The datas are pushed to the browser thanks to a GeoJson file. A Javascript script is called to get the datas from the Json file, theses data are turned it into Javascript variables in order to be processed by the web browser.

The interface has been designed to be simple and intuitive. The map fill seventy five percent of the web page, there is just a sidebar on the left. In order to have a nice and intuitive interface, we decided to use a template inspired by the Google material Design, thanks to the library CSS "Materialize" (http://http://materializecss.com/ , MIT licence). The API used to display the map is the "Google Map Javascript API" (https://developers.google.com/maps/documentation/javascript/).

8.3 How it works

When the user load the page, he is automatically geolocated thanks to a Javascript function. His position is displayed on the map and he can immediately see the points of interest around him. Note : as the data get from the datasets are only available for the U.S.A., the default position is not the geolocalisation of the user but it has been set up to New York. If the user wants to know other suggestions for another place, he can input a city name in the searchbar displayed on the map. This searchbar is based also on the Google Maps API. Suggestions and autocompletion are available for a better user experience. The user can also filter the activity displayed on the map. Theses activities are split in six categories, each category can be displayed or not easily by checking the corresponding checkbox in the sidebar. For every point of interest, a marker is pinned on the map. Each maker is clickable. On the click, more informations are displayed about the activity proposed to the user. Theses markers are generated thanks to the Google Maps API. Finally, the map inertia from the Google Maps features : the user can display the both 'Plan' or 'Satellite' view, and can zoom in and zoom out as much as he needs.

8.4 Notifications

Based on the forecast, five different kind of predictions are displayed to the user.Theses notification go from 'Storm' to 'Sunny day'. Here you can, see an example of notification (see figure 11).

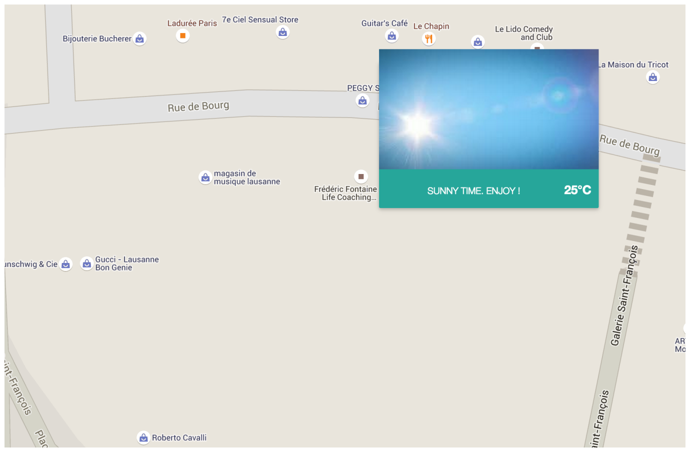


Figure 11: Web-application notification system

The notification is automatically displayed when the user load the page, and every time he decides to access forecasts for another city (using the search bar). The notification is displayed for three seconds and can also be swift by the user.

References

[1] Dr. R. Sluiter. Interpolation methods for climate data. 2008.