2.2.1 Using hashtags

Hashtags are probably one of the things that will remain Twitter's legacy, just as the @ method for addressing someone – e.g. @John is referring to John. Their use has spread from Twitter to Facebook and a few other networks. They give you the ability to filter out the data on a hashtag level, so it's essentially a twitter-specific keyword or a word of interest.

Here is a breakdown of a very simple hypothetical tweet:

@FunnyChap: Something witty and very well said **#jokeoftheweek**

In the tweet you can find the following:

1. The name of the user is denoted with a @ before the username.
2. The text itself.
3. The hashtag (a special Twitter construct) is denoted with a #.

Hashtags are one of the most popular ways of indexing and flagging up important words. That is why I thought that they will be a good fit for an initial investigation of whether the task is feasible. The fact that almost every major event and thing had its own hashtag was also quite nice, since you could then use that for TDT if you'd want.

The main thing that made me consider this option altogether is because the computation cost of finding those hashtags was minimal. Since they are already pre-processed and can be found in a separate attribute I didn't even had to look at the text. The implication of that was that I could easily reduce my dataset significantly, since I'll be throwing away most of the information. All in all the CSVs for every partition of my dataset are about 60 MBs, which is very small compared to the working set of 45 million tweets I had for the occurrences paired travel terms.

I was getting quite optimistic in thinking that could be a very simple and easy way to solve my problem and get the time series I needed. To my great disappointment it turned out that the volumes of obtained were quite small and overall the data was very noisy, which would have training a model very hard and predictions from them would've been quite bad.

In Figure 2.1 you can find a comparison of Hashtag counts and counts. They both have the spike in mid November, but the overall Local Polynomial Regression fit shows that the trends in both are quite different. The hashtags spike in February and the counts in November. That's quite a different behaviour and can result to very inaccurate predictions. Another very big caveat with hashtags is that even if a hashtag is related to a place, it's not guaranteed that the hashtag will have the place name in it. The #Olympics2014 was a trending topic for a week and is set in Sochi, however with the simple model there is no easy way to take that into account and handle it well.

The fact that they didn't work as my main Twitter feature does not mean I did not use them at all. They are incorporated in the 2nd version of the model (MultiFeatureTwitter) where I am looking at all the words that co occur with a particular place name and using those as additional features. I am going to discuss that in more detail in Chapter 3 on page 29 describing how that is used and why it could be beneficial to do so. To conclude - hashtags are useful not as a single input, but rather as an additional feature that can be used to enrich the model.

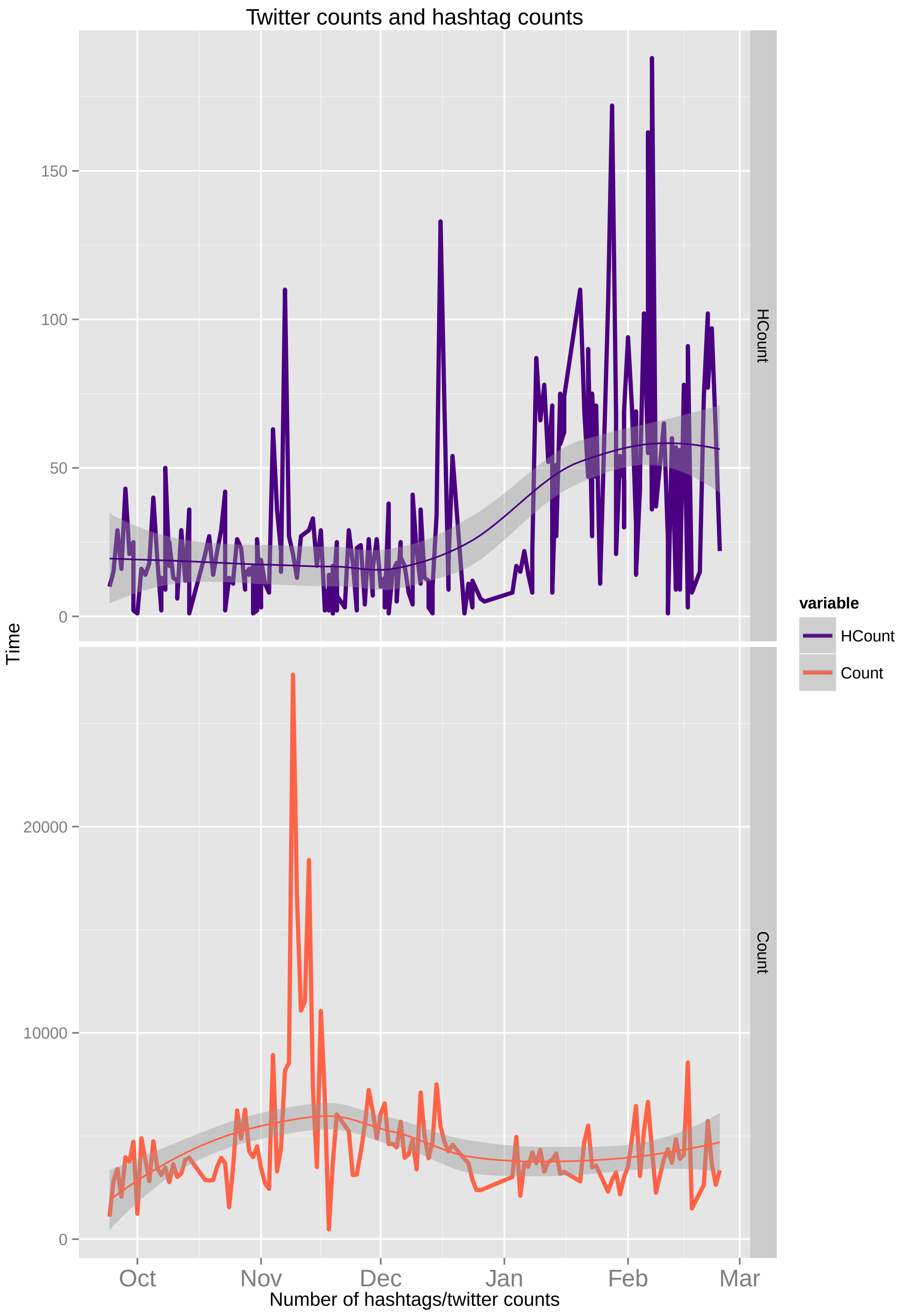

Figure 2.1: Hashtags (HCount) and Twitter counts (Count) for London. As we can see both of them have the spike in mid November, but with Hashtags it's quite hard to distinguish signal from noise. The Local Polynomial Regression fit (line) shows the overall trend.

2.2.2 Occurrences paired with travel terms

The simplest solution I could think of did not work as I had planned. Because of that I had to go back to the drawing board and think of another way to get everything I need from the data.

The next thing that seemed worth trying was to start using the actual tweet text and do some processing on it in order to get counts for the place time series. In terms of computation that was much more expensive, because even though the tweet text is limited to 140 characters, when you have 650 million tweets it turns into a non-trivial task.

In order to form the Twitter time series I excluded all the tweets that do not contains a place name and/or one of the travel words. In Figure 4.1 you can find all of the travel-related words which I used in order to perform the filtering. I picked them by trying to thing of all the words that someone would mention when going on holiday in any way – hence why “fly”, “land” and “trip” are all in the dictionary.

In my initial setup I had a few steps in order to get to the final time series:

1. Go through all of the data and take only the relevant tweets reducing the full set to a working set (around 10% of the overall).
2. Traverse the smaller set and generate the Twitter count time series for a destination – city or country.

I decided to partition the data into flat files by having approximately one day in every separate file. Initially everything was well, but after doing the processing a few times I realised that there was an error – every time there was a new data file I had to reprocess everything again. That wasn’t obvious at first, because while the dataset was less than 100GB the total working time was about 3-4 hours, however later it became obvious that this is a bottleneck since everything else depended on getting the data in a format that is easy to feed into the model.

In order to mitigate this I had to think of a better, more efficient and more scalable way of performing that step. My first decision was to combine the two steps into one. So instead of filtering first and traversing the dataset, I now traverse the full dataset and obtain the counts as I go through it. The working dataset is kept, because I need it to extract the additional features later.

Another drawback of the first version was that every time the process stops halfway through I had to re-run it completely, because I did not store any partial counts up to date. Instead, I was storing the full counts and outputting only after I’ve gone through absolutely every single file. In the second version I am using partial counts for every place and at the end of each iteration of the algorithm those partial counts were merged with the count up to the slice in order to produce the full counts later. That allows me to traverse the data in an easy way and the whole of it is done in about half a day, while with the previous solution it was taking more than 2 days.

You can find the pseudo-Python code in Figure 2.3. For the multiword place names I had to iterate the entire dictionary, because that is the best way. For the single word dictionary I perform lookups against the dictionary that contains them all. I have chosen dictionaries over lists, because amortised dictionary lookups are $\mathrm{O}\big(1\big)$, while list lookups are $\mathrm{O}\big(n\big)$. When you are doing over small lists and dictionaries there is no

```
terms = {
          "airport": "",
          "check-in": "",
          "fly": "",
          "land": "",
          "landing": "",
          "plane": "",
          "take off": "",
          "destination": "",
          "journey": "",
          "passenger": "",
          "route": "",
          "travel": "",
          "travel agent": "",
          "trip": "",
          "charger flight": "",
          "cruise": "",
          "excursion": "",
          "hostel": "",
          "hotel": "",
          "luggage": "",
          "motel": "",
          "package holiday": "",
          "self-catering holiday": "",
          "sightseeing": "",
          "go sightseeing": "",
          "suitcase": "",
          "tour": "",
          "tourism": "",
          "tourist": "",
          "vacation": "",
          "rail": "",
          "go by rail": "",
          "railway": "",
          "railway station": "",
          "road": "",
          "voyage": "",
          "board": "",
          "go on board": "",
        }
```

Figure 2.2: All the travel related terms I used in a dictionary. They were used as a very primitive additional filter to capture only the most relevant tweets.

```
one_word_placenames = obtain_one_word_places()
multiword_placenames = get_multiword()

counts = {}

for file in twitter_data_files:
    for line in file:
        tweet = parse_json(line)
        for word in travelwords:
            if word in tweet:
                travelWord = True
                break

        for i in multiword_placenames:
             if i in tweet:
                 seen_place = True
                 counts[place][datetime] += 1

        for token in tweet.split(' '): # split the text into words
            if token in one_word_placenames:
                seen_place = True
                counts[place][datetime] += 1

        if seen_place or travelWord:
            writeToFile(tweet)

for place in place_names:
     data = load_file(twitter_counts)
     partial_counts = counts[place]
     data = data.append(partial_counts)
     # group them by the date time and then sum them up
     data.groupby('Datetime')
     data.sum()
     output(data)
```

Figure 2.3: Pseudocode for filtering the data, outputting partial counts and then recombining them into the full time series.

Date	Twitter count
2013-09-25	X
2013-09-26	Y
2013-09-27	Z

Table 2.4: Twitter time series.

significant effect on performance, but in our case we are doing it for every line in every file therefore the reduction is much needed.

After the filtering and aggregation of the Twitter time series the working data set is 45 GB of data containing 45 million tweets. The reduced dataset is still much smaller in comparison to the full one. However eliminating the duplication of work made it much easier to re-process the data in a more efficient way and very easy feature extraction later.

After the processing we have simple time series for a city or country, which are simply the date and the Twitter counts for that date. In Table 2.4 you can see how the tabular files containing the Twitter counts look.

Overall, after doing the processing I have approximately 1500 files containing cities and countries time series extracted from Twitter. Those are then joined with the 2403 CSV files for the searches to those places. After the matching up we have 1377 files for 1376 cities and countries plus one for the overall search and tweet volumes. Each of those files has information on the searches to that place for the day and how many times it was mentioned on Twitter.

2.3 Exploring the correlation between the time series

Since now we have both of the time series joined up, it was time to see whether the two time series are correlated. There are several ways which allow one to explore the relationship between two variables.

The method I chose to do that was the most logical – carry out the Pearson test between the search volumes and the Twitter counts in order to find out what is the relationship in the two. After carrying out the test for all the destinations the results for some were somewhat disappointing. In Table 2.5 you will find the top and worst 10 destinations by absolute R value, which allows us to see which ones are correlated and which ones not.

All in all, 1286 of the places had an R value of less than 0.2 and the remaining 91 have a value of > 0.2. A correlation of such small magnitude is not even worth reporting, however, since my task it not solely defined by predicting one attribute from another I wanted to include in order to give a perspective on the difficulty of the task. In Figures 2.4 to 2.7 I have shown 4 examples of particularly interesting destinations.

The best correlation is for Sochi, which has a correlation coefficient of 0.78. For Sochi we can observe at the top right a cluster of dates which have high volumes of tweets, but also a proportionally high number of searches. As we will see in Table 4.9 on page 49 quite a few of the features picked out by the model were to do with the olympics – both the positive and the negative aspect.

Place	**P value**	**R value**	**ABS R value**
Sochi	0.00	0.78	0.78
Manaus	0.00	0.56	0.56
Salamanca	0.00	-0.47	0.47
Najaf	0.00	0.47	0.47
North Korea	0.00	0.46	0.46
Marshall Islands	0.00	0.46	0.46
Faisalabad	0.00	0.42	0.42
Fukuoka	0.00	0.42	0.42
Uruguay	0.00	0.42	0.42
Ukraine	0.00	-0.36	0.36
Des Moines	1.00	0.00	0.00
Parma	1.00	0.00	0.00
Burundi	1.00	0.00	0.00
Quito	1.00	0.00	0.00
Vieques	1.00	0.00	0.00
Algiers	0.99	0.00	0.00
Lampedusa	0.99	0.00	0.00
La Coruna	0.99	0.00	0.00
United Kingdom	0.99	0.00	0.00
Tamworth	0.99	0.00	0.00

Table 2.5: Top 10 and worst 10 destination by absolute R value. Sochi and North Korea have a strong positive correlation, while Ukraine is very negative. As you can see for the worst 10 see it the P value for all of them is nearly 1, which means that for these place any uncorrelated system could produce very similar results as those.

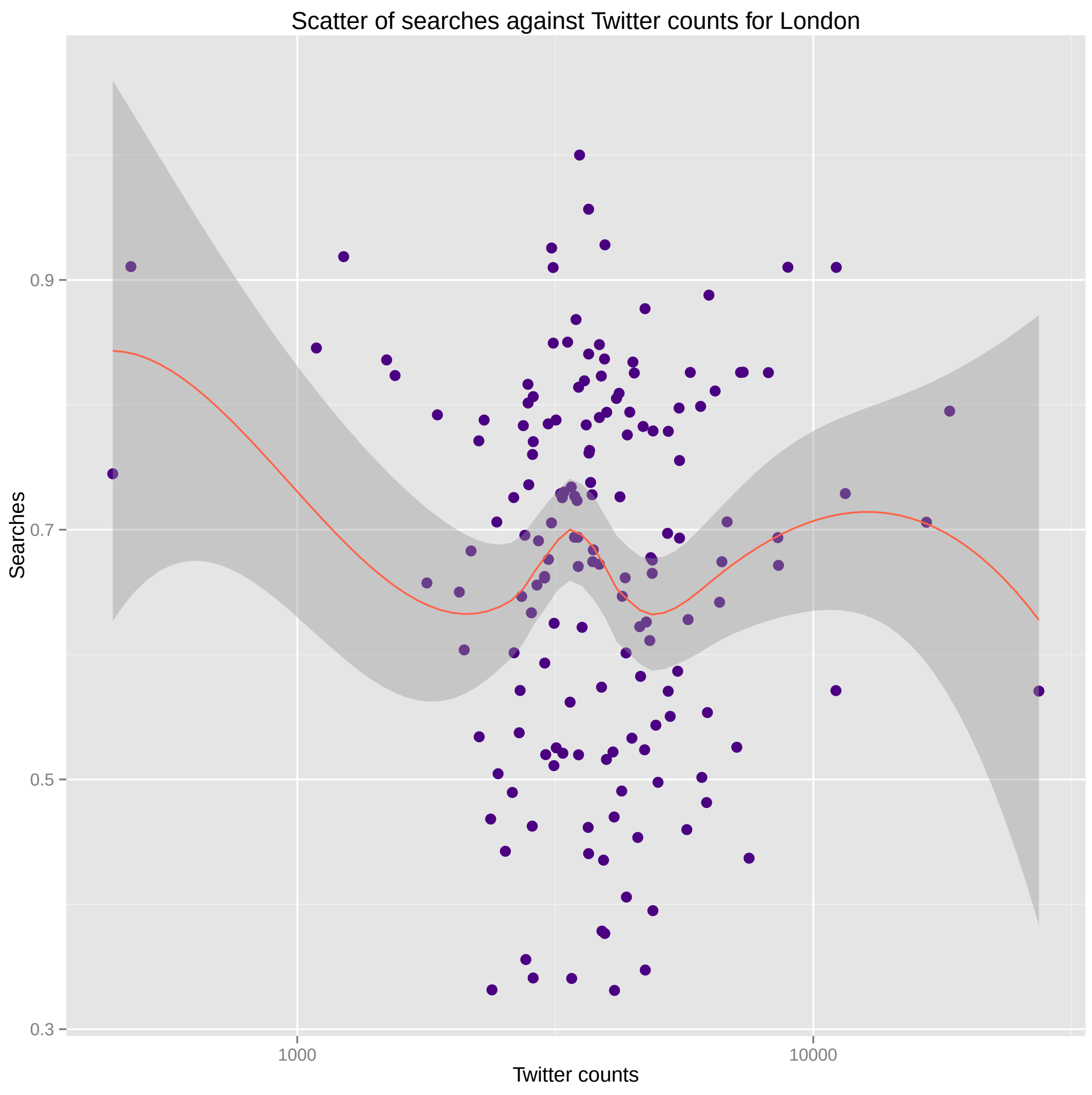

Figure 2.4: For London the correlation coefficient is 0.003. We can see that there is no clear pattern and the P value implies that two random systems that are not related at all can produce similar result.

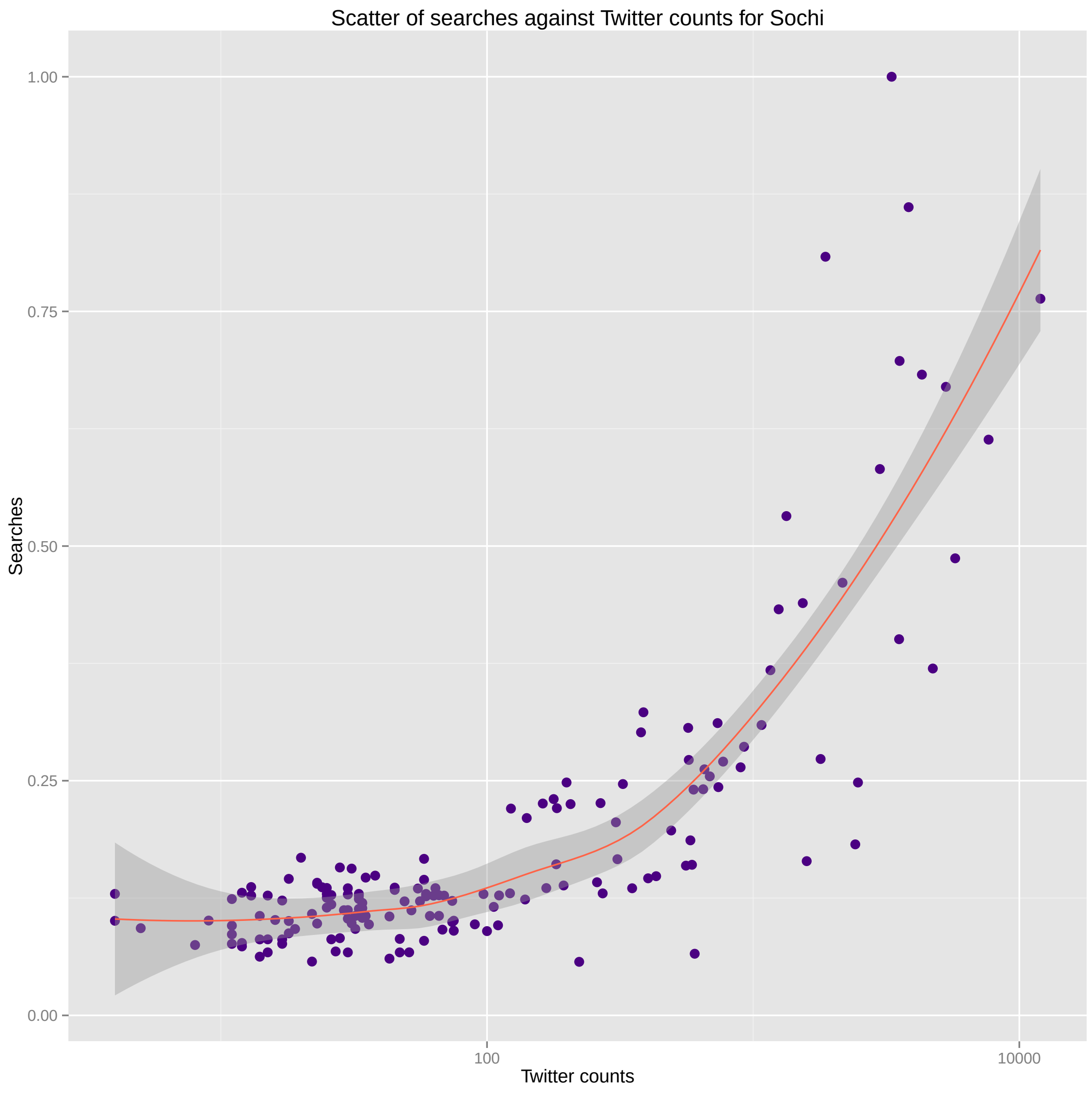

Figure 2.5: The Olympiad has bumped the correlation coefficient for Sochi up to 0.77, because people were tweeting and wanted to go and see the Olympiad.

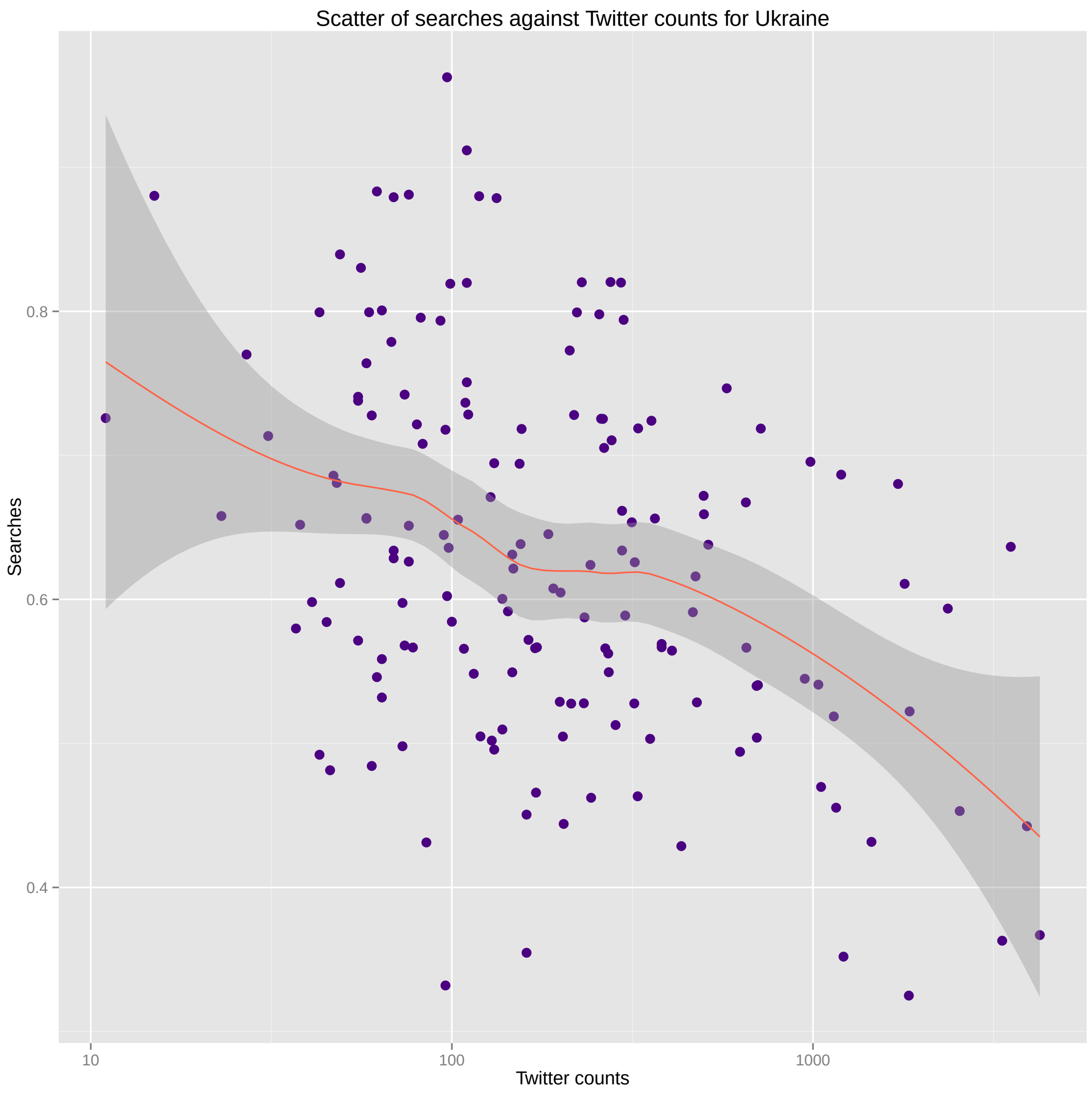

Figure 2.6: Ukraine has a correlation coefficient of -0.35. This is interesting, because the recent protests decreased the number of searches even though people were tweeting a lot, therefore the recent events made them less likely to want to go there.

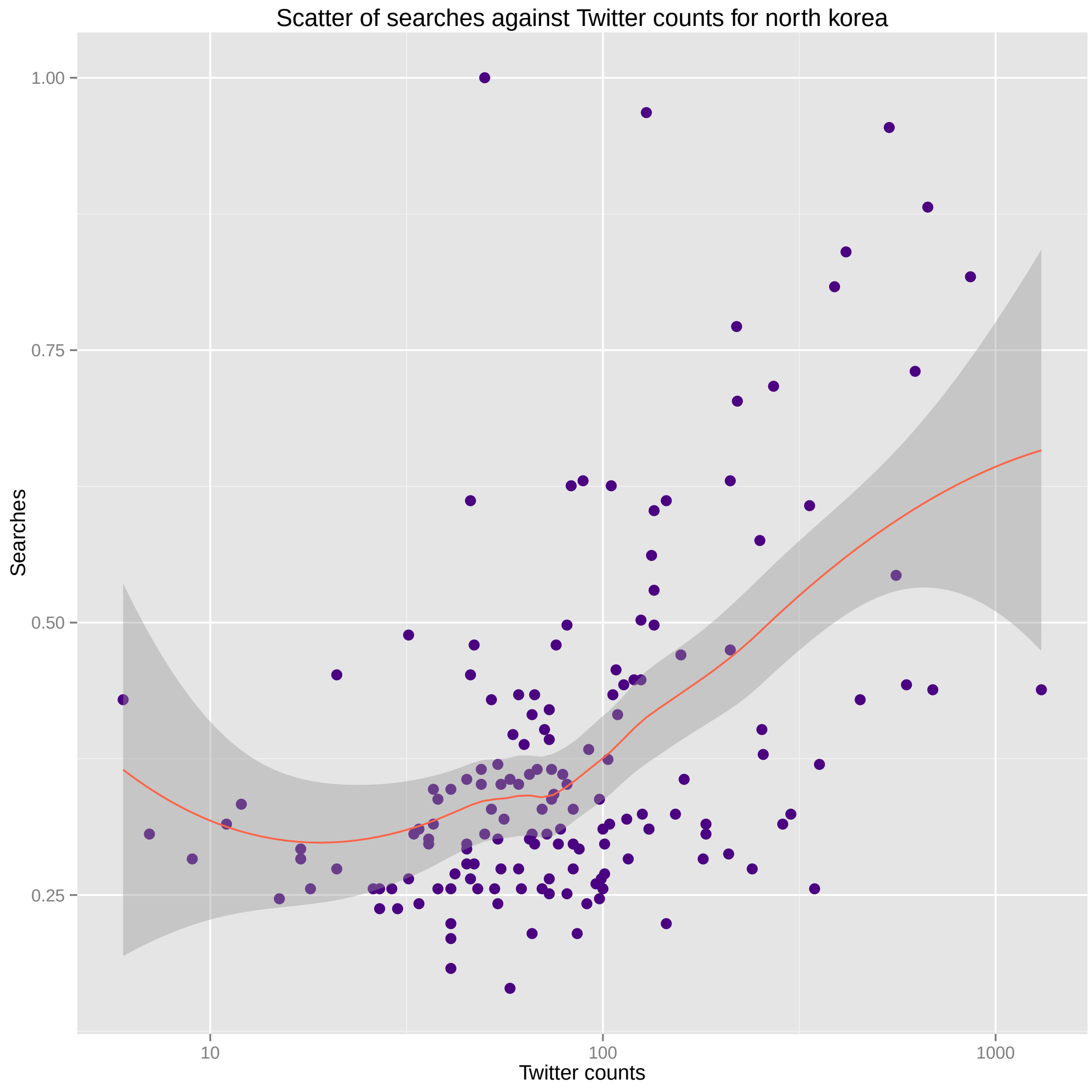

Figure 2.7: With a coefficient of 0.45 North Korea is an interesting outlier. Tourism to North Korea is quite a niche market and even though the sentiment is usually negative, people want to go there nonetheless.

Puzzling for most North Korea has got a similar profile, even though the correlation is not as strong. With a correlation of 0.46 it is the 5th best correlation altogether. Tours to North Korea do exist and therefore every time North Korea is mentioned in the news people try to see whether there are flights to that curious place on Earth.

The correlation for Ukraine is very negative and so is for Kiev, both caused because of the recent problems in the country. What is really interesting would be to try to determine what causes the correlation to improve or what makes it worse. I did not have much time to work on that, but using some NLP techniques I believe it is entirely possible to be even more descriptive in what are the exact factors that influence it.

London on the other hand has a very small correlation coefficient and it is expected that the Twitter models will not perform as good, since the two variable are unrelated and Twitter is not going to play much part in the prediction.

For those who have a value higher than 0.2 there is definitely some correlation, which could be pushed lower by all the outliers and holes in the data. Fixing that requires some cleansing of those holes in the data. I've described everything that I did for the cleansing in Section 2.4.

These results mean that there is a knock-on impact on the models that need to be used in order to perform the prediction. I wouldn't be able to fit a linear regression, because that will give predictions that are very far from accurate. It also means that when doing the model I wouldn't be able to make one that works just with Twitter or the Twitter features.

A hybrid method would be needed. We need to be using past search data as a component as well. That will ensure that weights of all the features are balanced. That is because the initial idea is to use Twitter as an exogenous factor with some weight. The best model that could do that job is LASSO [23]. Due to the very fact that every place is quite different - apart from those with no correlation whatsoever - I fitted a model for each one and one for the overall searches and tweets, which can be found in more detail in Chapter 3 on page 29.

2.4 Data cleansing

As I have previously mentioned there are two data sources which I used for this project - data obtained from the Twitter Streaming API and the Skyscanner search volumes. Even with one you can always experience some outages and problems. In this project I used two, therefore there are a couple of sources of risk:

1. The script that collects data from the Streaming API - sometimes the API just wouldn't work.

2. The searches data coming from Skyscanner being incorrect or partial - not spanning the full date range.

The dates with incomplete or missing data can impact the regression negatively so it was vital to tidy up all of the data by either backfilling it or filling/removing the missing values.

As far as the Skyscanner dataset is concerned I have backfilled of data missing by using the Last 4 Fridays, method described in detail in Section 3.1, in order to ensure that there are enough data point for carrying out all the tests. For the twitter counts, I have simply taken the mean for the values of missing Skyscanner data and dropped all the days where the Twitter collection had problems. For the full features I've just filled the missing values with 0, since there wasn't any better of filling those without removing all the hashtags and words that appear around a specific time and then disappear again.

2.5 Feature extraction

With the whole dataset prepared for the statistical analysis and model I wanted to prepare additional data, which was going to enrich the prediction and perhaps make the model a bit more robust, since now it was going to incorporate more features. With those features one can easily do a better deep dive by re-training the models in order to see which features are the most helpful in terms of bettering the prediction. I thought that there is also going to be some data on what events were happening during that time, because quite a few of the feature are not going to be present for absolutely day – the Olympics for instance was trending around the event and significantly improved the correlation for Sochi.

As an additional note, I am not taking *absolutely* everything that co-occurs. Since humans are fallible and Twitter is a place where everything can be found, I had to remove some weird characters which are used to form the basis of what is called "ASCII art" and all the different "emojis", which are basically emoticons.

This was a computationally intensive task and probably the most difficult bit to get up and running without any major hiccups. What I did for this particular part of the project was something quite similar to what is shown in Figure 2.2, however instead of just adding to the count of a place, I add to the counts of all the words that co-occur with the place name. Pseudocode of the way it works is shown in Figure 2.8. The algorithm is simple enough, but the fact that the dimensions of the count dictionary are massive I had to be extremely careful not to load up too many of the processed files for extraction at once, because that lead to some memory errors.As a result I decided to split them into partial counts and recombine those.

Just in terms of numbers - for London I had 148877 features. The CSV file for London had 148877 (features) times 160 rows (days). It wasn't very easy working with the big sets, because in order to perform all of the predictions they all need to be held into memory, but thanks to Pandas [24] the fitting and evaluation of those extended models takes about 2 hours, which is 3 times less than 6 hours without using CSVs and pandas data frames.

```
place_names = get_all_placenames()

counts = {}
# Obtain all the counts
for file in twitter_data_files:
    for line in file:
        tweet = parse_json(line)

        for place in place_names:
            if place in tweet:
                # split the text into words
                for token in tweet.split(' '):
                    if token in one_word_placenames:
                        seen_place = True
                        counts[place][token][datetime] += 1

        if seen_place or travelWord:
            writeToFile(tweet)

# Recombination and output the final result

for place in place_names:
     data = load_file(feature_file)
     partial_counts = counts[place]
     data = data.append(partial_counts)
     # group them by the date time and then sum them up
     data.groupby('Datetime')
     data.sum()
     output(data)
```

Figure 2.8: Pseudocode for the feature extraction task.

3.4 MultiFeatureTwitterDF and CF

Since the simpler TwitterDF and TwitterCF worked really well, I decided to continue exploring and create a more sophisticated model. In order to do that I used the features which I acquired by performing the data processing described in Chapter 2.5 on page 26. For the top 80 cities/countries and 2 additional ones – Ukraine and North Korea – I took absolutely all of the words that co-occur with those cities/countries and used them as additional feature. That of course mean that for some places I had more than 100 000 features to work with.

The massive number of features was not a problem at all for LASSO. I've had a set of values of α – 0.5, 1, 2,5, 10, 20, 50, 125, 250, 500, 1000, 2000, 4000, 8000, 16000, 32000. Since the more you increase the value of the parameter, the more junk weights you throw away, the RMSE proportional to the number of features with nonzero weights. Even though 32000 is a very big value for the parameter, for some destinations with big number of features it's worth going even further beyond that to reduce the weights to less than 10.

Figure 3.1shows the improvement in RMSE with reduction of number of non-zero weights as alpha increases plotted against the values for α for all two MultiFeature models. As you can see there is a very good correlation between the increase in value of alpha and the reduction of the two plotted things.

In Chapter 4.3 you will find a detailed summary of the results and discussion for the MultiFeatureTwtter models. I have picked several interesting destinations which were the best to show the gains and losses in using the expanded feature set when trying to improve prediction. The model gave mixed results - it improved predictions for some things, but all in all it did not perform better than the simple TwitterCF and TwitterDF.

3.5 Model for all the searches

Since we have got many so many different models for every city and country in the world I also decided to investigate what is the benefit of having one single overarching model. As a result the prediction for the overall search volumes was improved by 3.1%. The summary of the weights can be found in Table 3.3. It's quite useful having a model for the overall volumes as well because that gives a very quick overview of what is happening.

Twitter	F1	F2	F3	F4
1.00	0.65	0.003	-0.06	0.13

Table 3.3: Summary of weights for the overall model

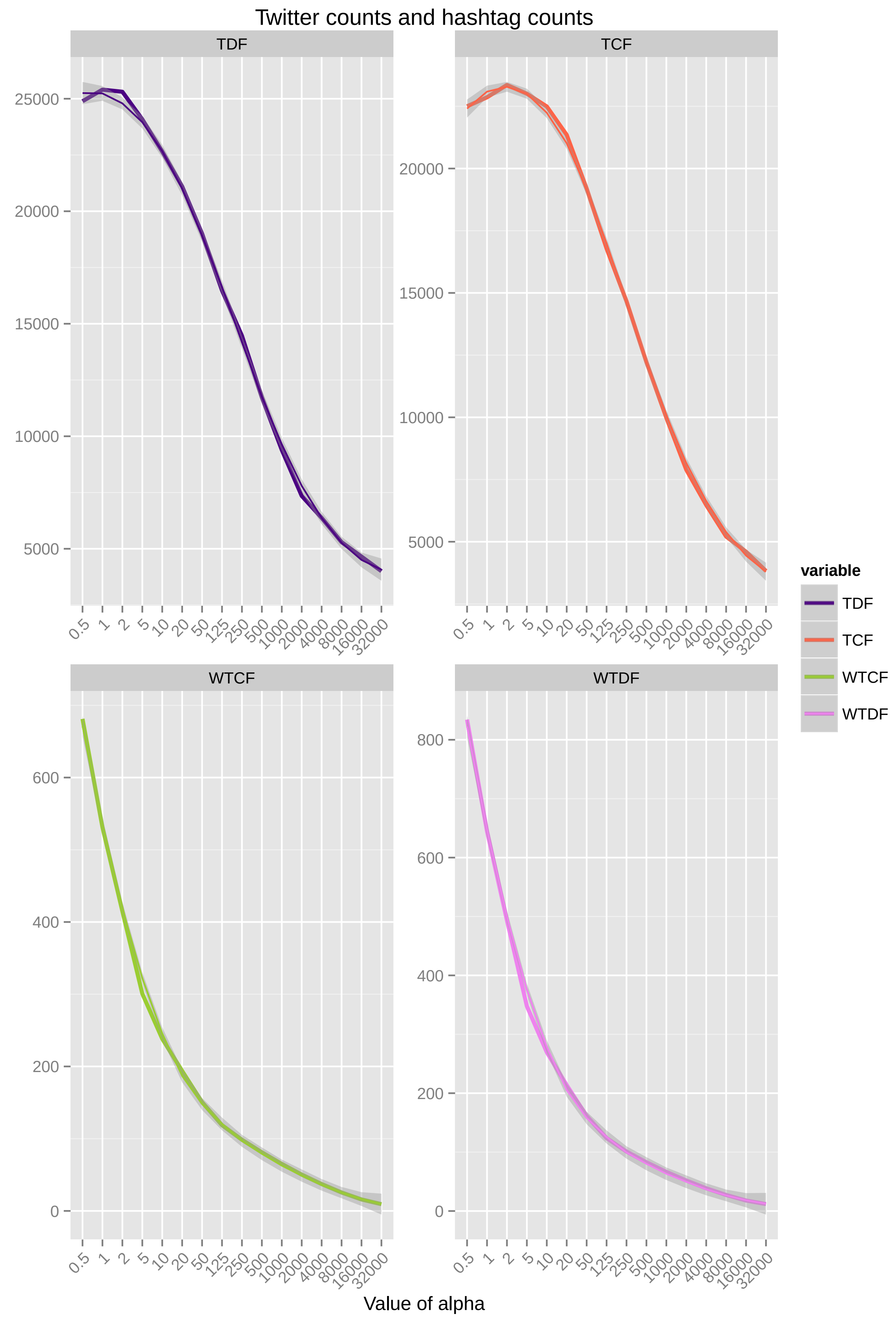

Figure 3.1: Reduction of RMSE and non-zero weights as alpha increases. TCF and TDF are the RMSE for MultiFeatureTwitterCF and MultiFeatureTwitterDF respectively. WTCF and WTDF are the number of non-zero weights for the two classifier. The results are means of all the actual values across all models.

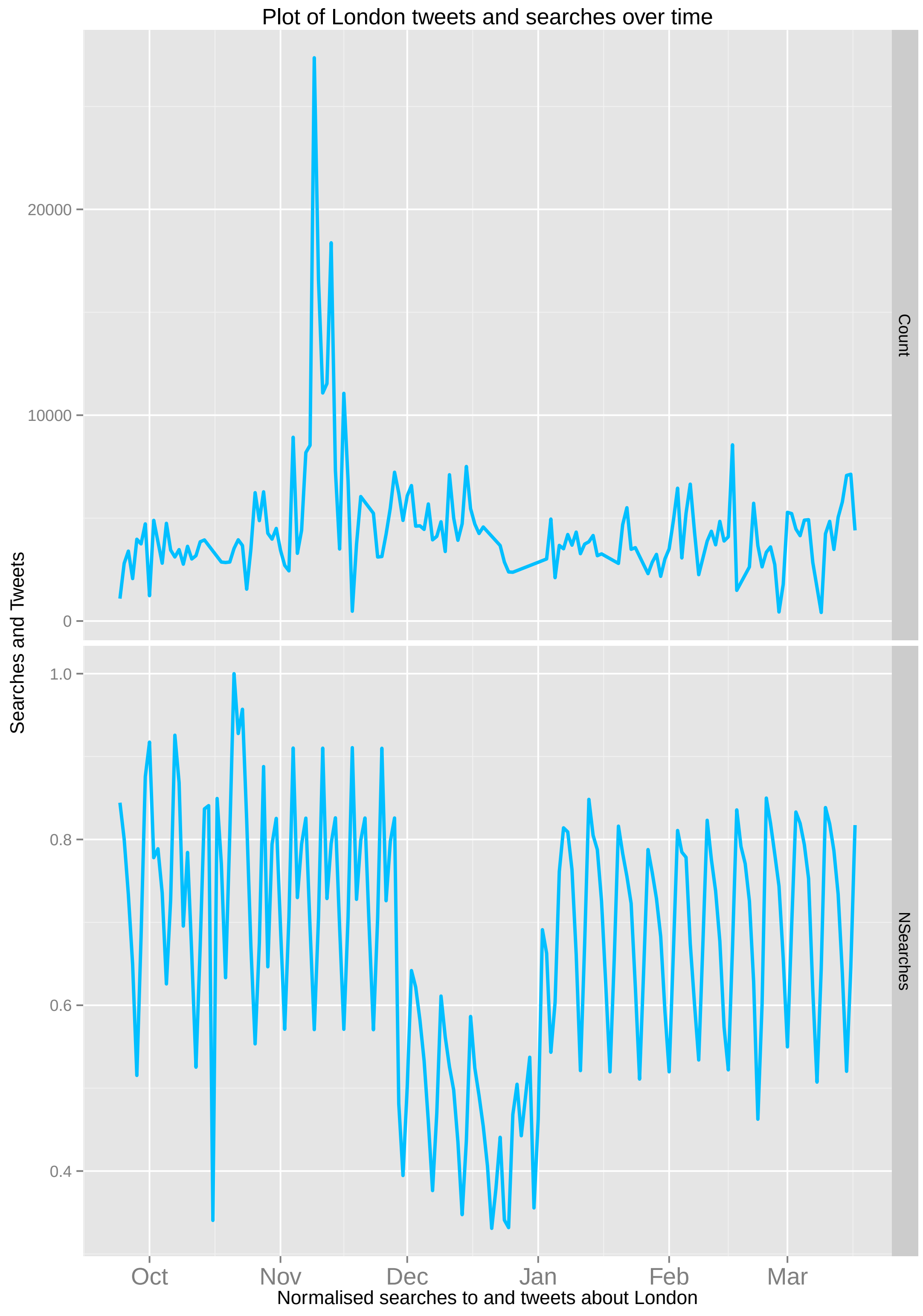

Figure 4.1: Searches to and tweets about London - as we can see the trend line is going to be almost parallel to the X axis.The slight dip in December is due to seasonality. That is noticed across the industry as a whole

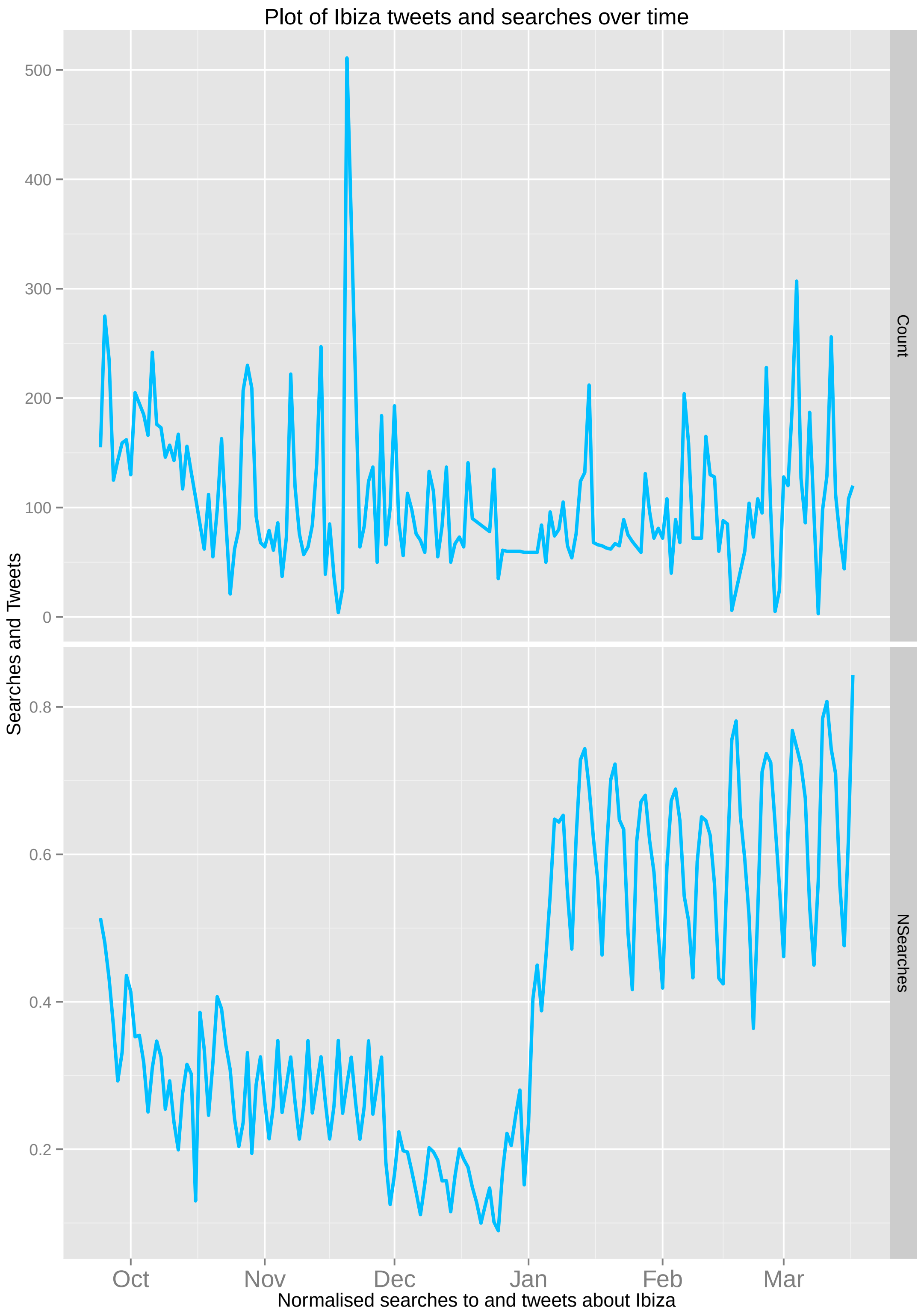

Figure 4.2: Searches to and tweets about Ibiza - in this plot we can get the idea of seasonality. The dips is observed as soon as enter autumn and then in January the searches to such destinations pick up again as people are starting to plan their summer holidays

4.1 Results for TwitterDF

Obtaining the initial set of results is vital for any project. In order to get to that stage I had to have working model/s which can perform the task of predicting search volumes with the help of Twitter. As described in Chapter 3.2, I picked LASSO for all the feature selection and regressions.

TwitterDF stands for Dynamic Fridays, which basically means that the weights for the past days are not predetermined as the ones in the L4F exponential weighting scheme, but instead they are automatically picked by LASSO. This way we get an incremental improvement over L4F, but in the model taking into account one of the many exogenous factors that could influence the search volumes either upwards or downwards. I expected that to yield better results for the 2nd group of destinations and possibly for some from group 1.

Overall, I was very pleasantly surprised by the results. On the full set of 1372 destinations which had non-zero RMSE, the first iteration of my model generated better predictions and had lower RMSE on 1034 of the destinations. On the other hand the in-house champion Last 4 Fridays performed better on 338 of the destinations. In order to illustrate that difference there is a scatter plot of the RMSEs in Figure 4.3. That means that my model is better on 75.3% of all the destinations. In the next two runs of training and evaluation, my classifier did better on 74.5% and 75.6% of all the destinations. After re-running the experiments, multiple times I am convinced of the significance of the results.

In Figure 4.3 we can see that the performance of the models is actually quite similar. Almost all of the dots are distributed along the x axis. Contrary to my expectations that destinations from group 2 would be better classified by my model, actually it performs just as well on the more constant ones from group 1.

In Tables 4.1 and 4.2 you can find the places that had the worst performing models and best performing ones in order to give an illustration of what is the improvement and degradation in performance. As we can see in Table 4.1 the destinations here are very small niche destinations, which are very unpopular, so when something happens people are more likely to search to flights to these places. In Table 4.2 we can see something quite similar - overall the decrease for some quite small places with Venezuela being the outlier. Due to the protests the correlation has become very negative and because of that even though people mention it a lot, no one wants to go there, since there is great unrest in the country.

If we look at the top 10 destinations in terms of RMSE (and in flight searches) in Table 4.3 we see something quite surprising. The improvement is positive in 9 out of the 10 examples. Even though it's small, the very fact that there is an improvement, means that taking into account exogenous factors is beneficial and can contribute positively to the quality of the prediction.

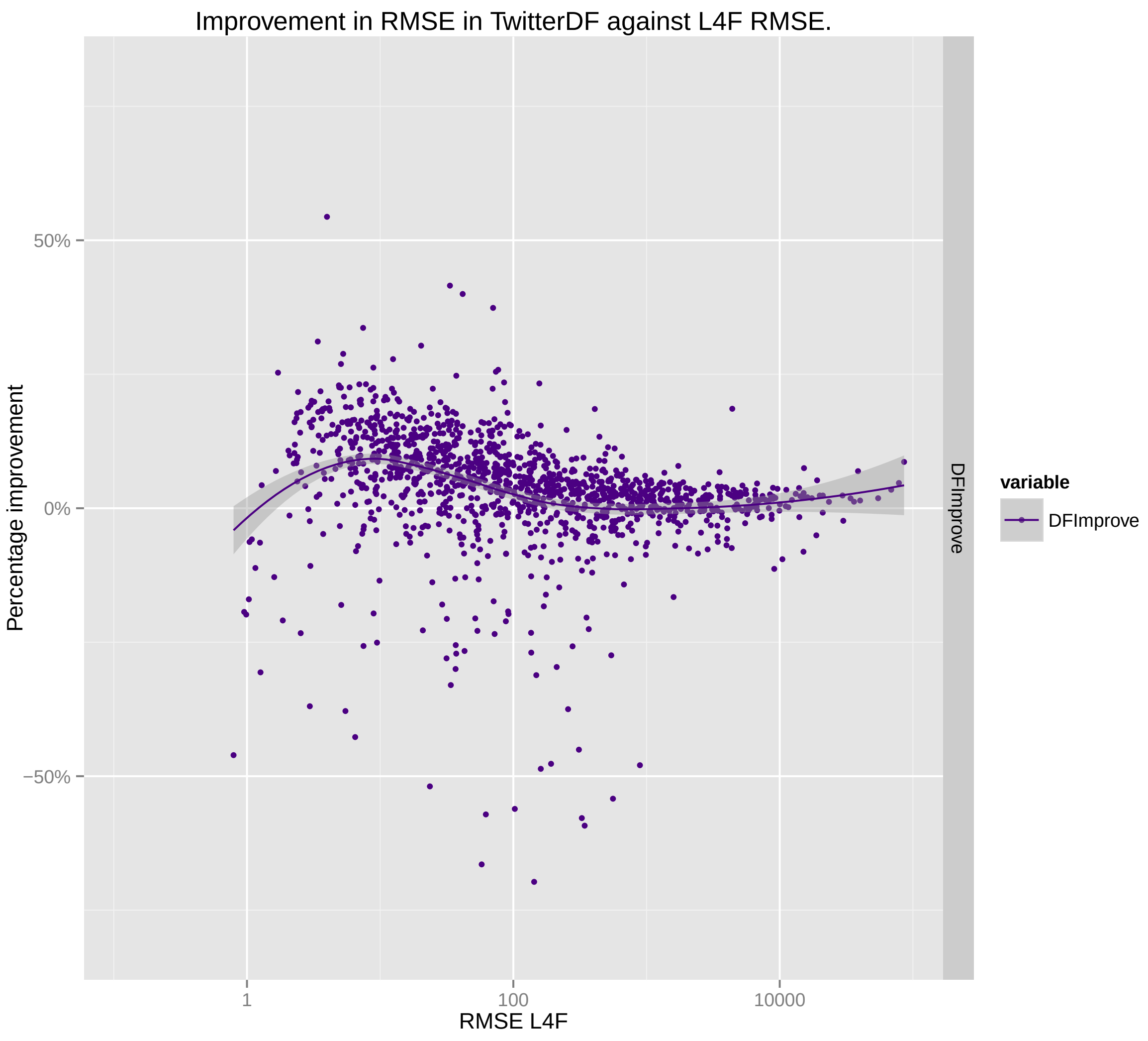

Figure 4.3: Scatter plot of the percentage improvement from the TwitterDF model over the baseline L4F. To put things into perspective, I've also plotted the y=x line to make it easier to visualise the border between the two. Anything below the line is where my model performs better and anything above L4F.

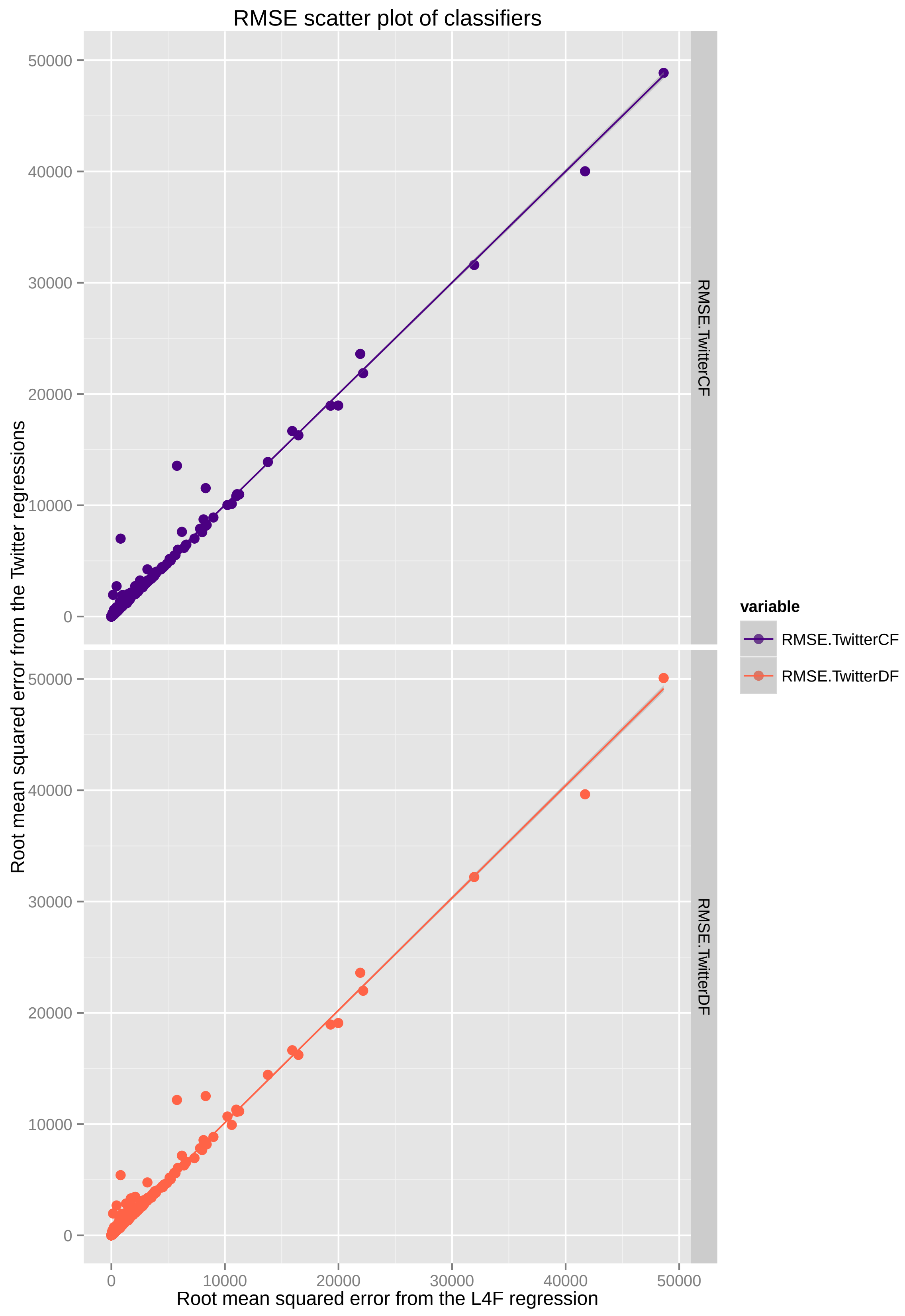

Figure 4.4: The two models compared: TwitterCF stands for compound Friday – taking all of the previous Fridays with predetermined weights and using them as a single weight in the regression. TwitterDF is leaving everything to LASSO.

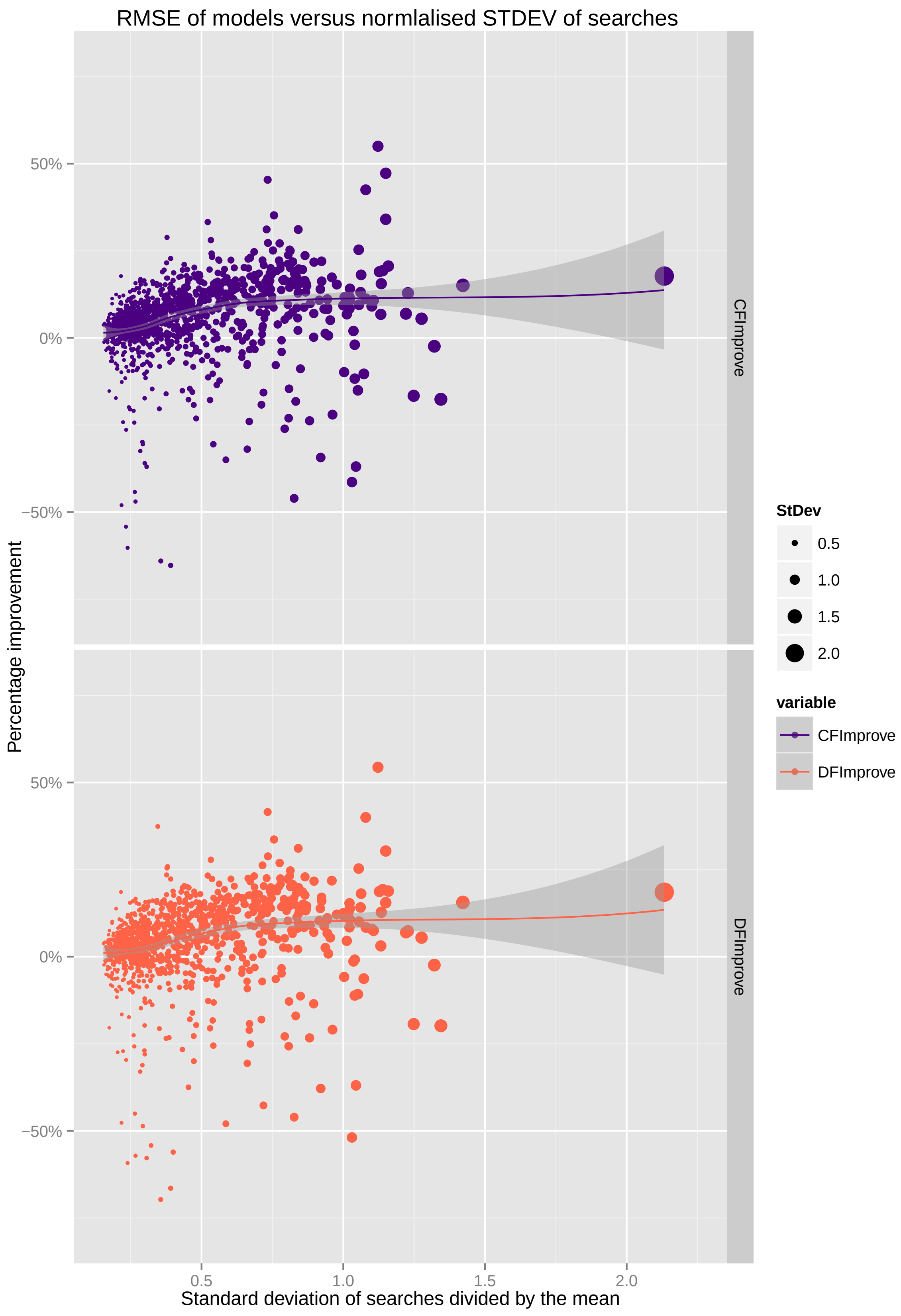

Figure 4.5: Scatter plot of normalised standard deviation (st. dev. divided by the mean) for the number of searches plotted against the percentage improvement for the two classifier over the L4F. On the left is TwitterCF and on the right one can find the improvement for TwitterDF. It seems that the Twitter models are doing good across the whole board.

α	> 0 W CF	MultiTwitterCF	> 0 W DF	MultiTwitterDF	L4F	Best
0.50	191	11,683	200	14,961	1,517	L4F
1	175	12,507	177	15,647	1,517	L4F
2	146	13,288	151	15,161	1,517	L4F
5	136	13,607	131	14,387	1,517	L4F
10	116	12,728	114	11,341	1,517	L4F
20	112	11,379	108	8,950	1,517	L4F
50	88	6,601	85	5,197	1,517	L4F
125	51	1,946	56	2,009	1,517	L4F
250	33	1,684	36	1,788	1,517	L4F
500	22	1,596	23	1,465	1,517	TDF
1,000	11	1,423	13	1,593	1,517	TCF
2,000	5	1,356	8	1,815	1,517	TCF
4,000	1	1,366	4	1,854	1,517	TCF
8,000	1	1,366	4	1,853	1,517	TCF
16,000	1	1,366	4	1,852	1,517	TCF
32,000	1	1,366	4	1,850	1,517	TCF

Table 4.6: Improvement in RMSE for Tenerife. As we can see after we get to a certain point the Twitter models get better and better, until it gets to discarding all of the weights, but the Fridays. The optimal result for TwitterCF is with 5 weights and for TwitterDF – 8, since the reduction in RMSE afterward is negligible, while the DF version performs best with 23 and it starts getting worse. The features and weights at $\alpha = 1000$ are in Table 4.6.

Feature	Weight
time	583.35
#tenerife	-87.43
#solareclipse	35.74
tenerife	-19.29
friend	64.81
Fridays	0.75
begins	-30.50
heres	7.72
would	-273.96
canary	-7.56
view	-10.86

Table 4.7: Weights for Tenerife at $\alpha = 1000$. These are the weights that the TwitterCF model has picked.

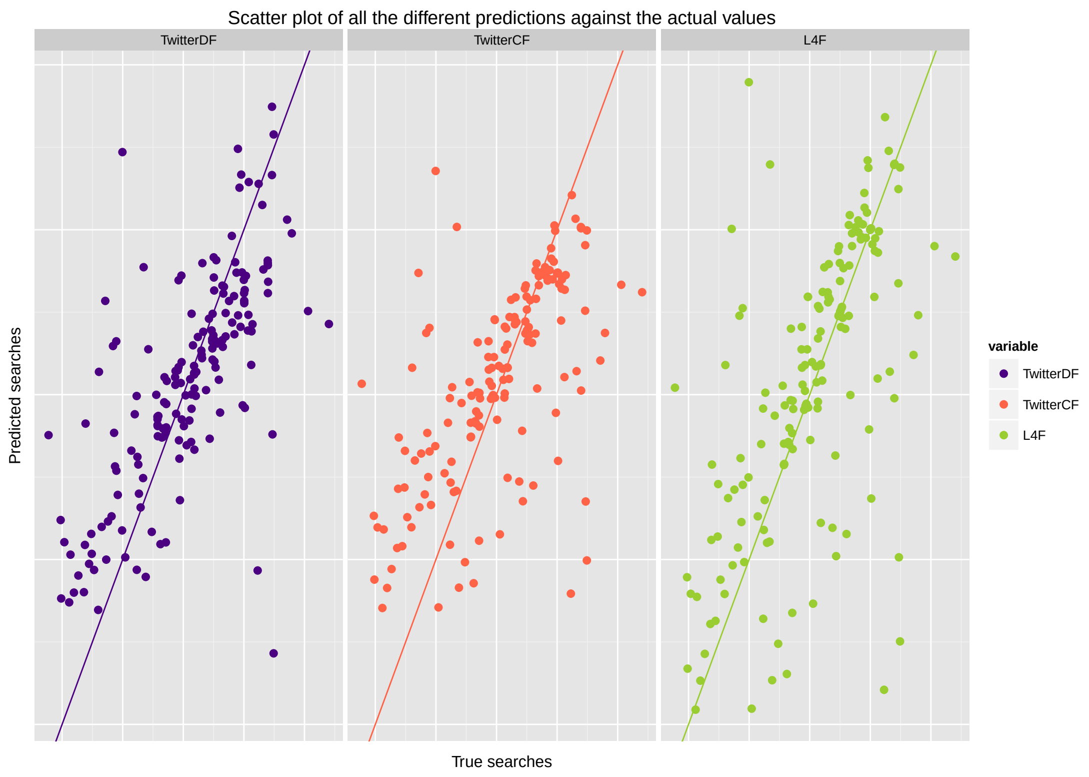

Figure 4.6: Predictions vs actual for a destination from group 2 - Tenerife. For this group Twitter models make actual gains, albeit small. The line $y = x$ is shown to portray the quality of the prediction.

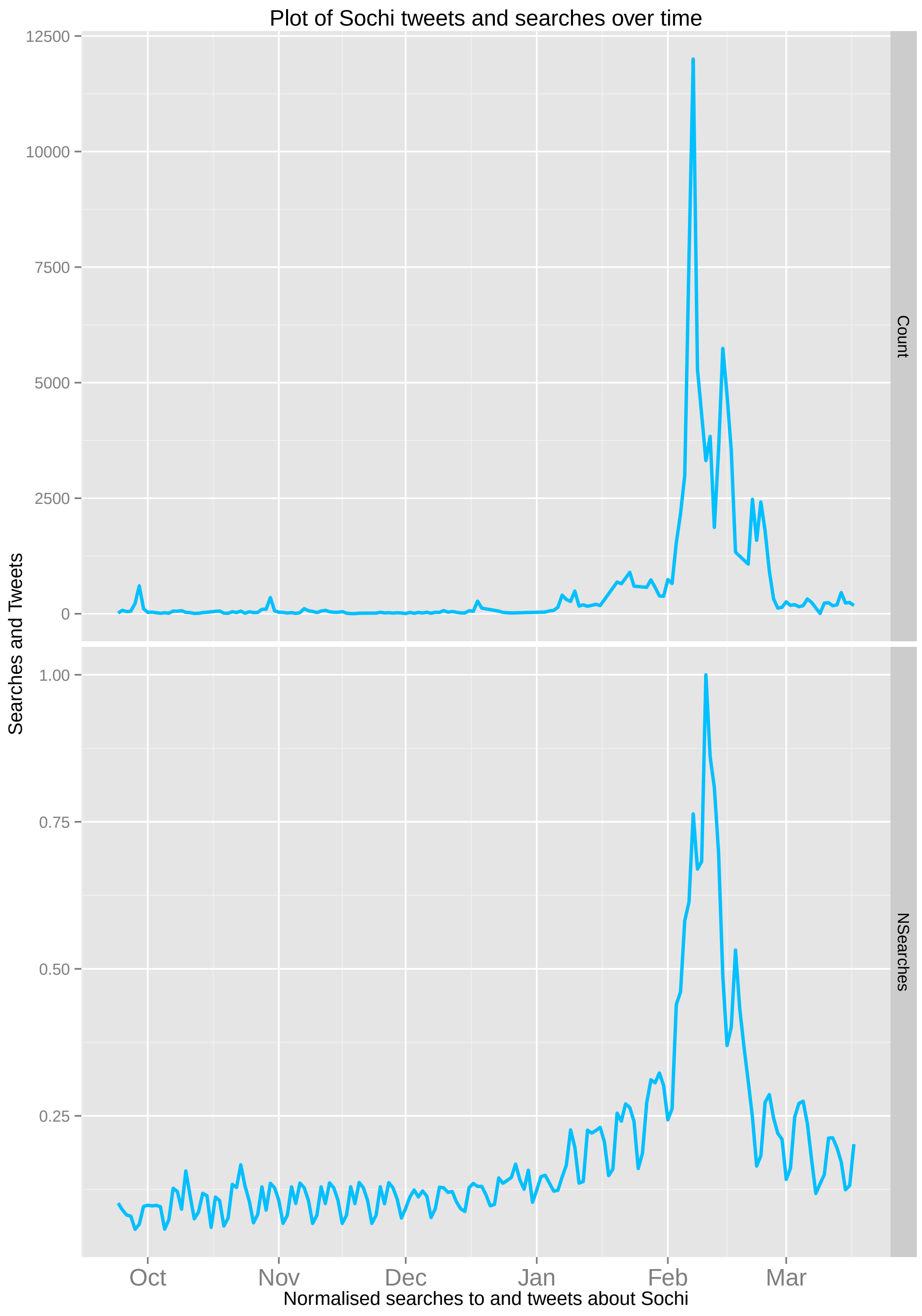

Figure 4.7: Searches and tweets Sochi. The massive spike is the Olympics. As we can see the two time series are correlated and the spike it tweets is just a bit before the spike in searches.

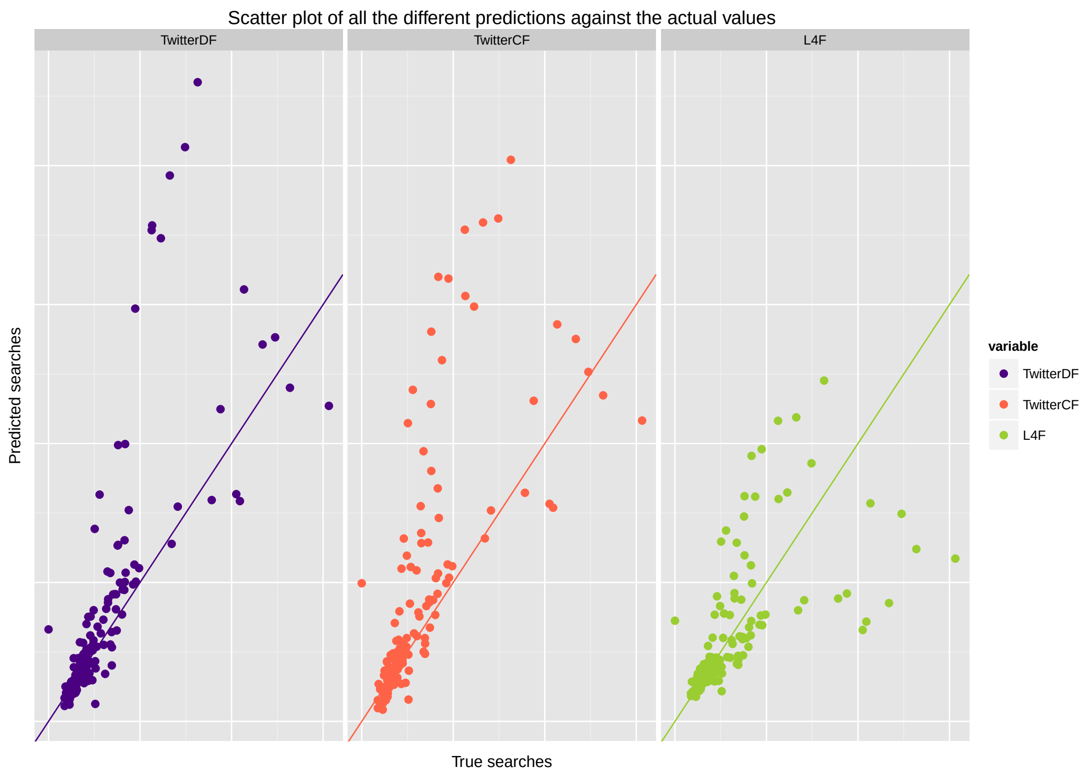

Figure 4.8: Predictions vs actual for a destination for Sochi. The olympics made any pattern in the data very hard to spot. The line $y = x$ is shown to portray the quality of the prediction.

4.3.3 UK

The previous two examples showed that a for some destinations there is definitely an improvement by taking the expanded set of features, But also it not only improves the predictions, but by looking at the most prominent weights, we can infer some information on why these features are picked – they can relate to events or could reflect on the socio-political situation.

The United Kingdom is definitely not one of the destinations, where we see an improvement as shown in Table 4.10. Even though there is a very significant improvement in the RMSE as we increase the regularisation parameter it is still not enough to bear the baseline model. At $\alpha = 32000$ there are still 14 and 11 weights in TwitterDF and TwitterCF. That means that if we wanted we could continue increasing the penalisation factor further, but that would turn tweaking and tuning the classifiers into a very complicated and laborious task and of course we will get into the problem of overfitting.

α	> 0 W CF	MultiTwitterCF	> 0 W DF	MultiTwitterDF	L4F	Best
0.5	583	60,924	649	70,068	8,700	L4F
1	467	65,234	523	68,728	8,700	L4F
2	369	65,253	412	76,945	8,700	L4F
5	280	59,667	290	69,158	8,700	L4F
10	231	54,972	241	62,057	8,700	L4F
20	197	58,161	212	55,629	8,700	L4F
50	167	62,001	175	77,956	8,700	L4F
125	147	59,146	155	90,270	8,700	L4F
250	139	73,864	142	105,197	8,700	L4F
500	116	81,806	120	96,173	8,700	L4F
1,000	106	84,796	110	91,576	8,700	L4F
2,000	98	61,891	94	60,692	8,700	L4F
4,000	71	44,306	68	61,037	8,700	L4F
8,000	44	56,021	43	49,030	8,700	L4F
16,000	27	31,209	29	28,339	8,700	L4F
32,000	11	15,895	14	16,034	8,700	L4F

Table 4.10: Expanded features for the United Kingdom. We don't notice the same gain here. Both TwitterDF and CF decrease the number of features significantly the RMSE for both is about 2x times the one for L4F.

4.4 Results on the full dataset

Another interesting benchmark was how it performs on the overall search volumes. In Table 4.11 you will find how the two models perform on the all the Twitter counts summed together and the searches to all of those destinations.

	TwitterDF	Twitter CF	L4F	Difference
Overall	1,640,111	1,621,391	1,674,130	3.1 %

Table 4.11: RMSE on the tweet counts for all destinations and searches to all destinations. Difference is the percentage improvement of the best of the twitter models.

As you can see in Figure 4.9 the overall volume of searches is not as volatile as the destinations in group 2, so it would fall into group 1, but surprisingly enough the Twitter models do a marginally better job at predicting.

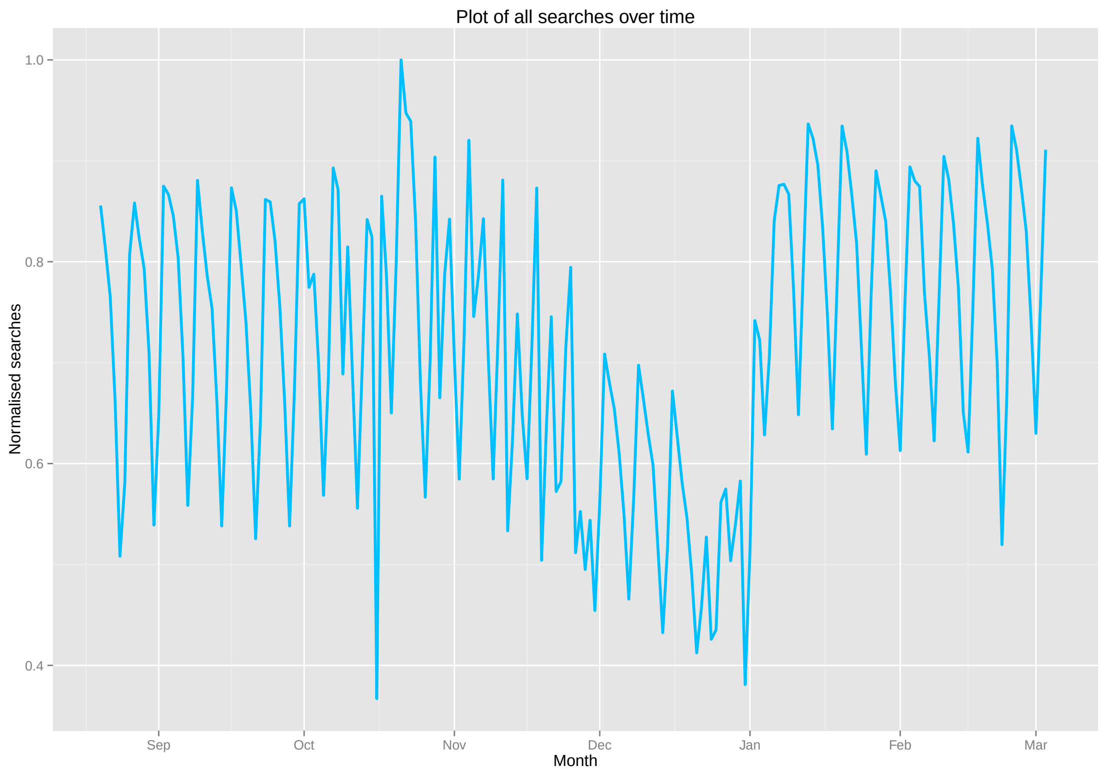

Figure 4.9: Plot of all the searches to all the destinations. This would fall into the first group of steady constant volumes