*December 08, 2013*

GMU, Fall 2013, CS 780 Data Mining for Multimedia Data, Prof Jessica Lin
Class Project Report
By
Talha Oz and Venkat Tadakamalla

# Partly Sunny with Chance of Hash Tags

*a Kaggle competition*

## Abstract

Human race for the first time in the history has huge amount of ready to compute publicly available textual social data, some of which is microblog posts such as twitter's tweets. If we have a better understanding of this microblog posts, it may help service providers to serve the society better. This study examines data mining, machine learning and computational linguistics methods to learn sentimental, temporal and semantic information from microblog posts to predict a multiclass multi-label (continuous value) problem. Our case study is on predicting human evaluation of weather related posts, as known as 'tweets' on Twitter. Various preprocessing, feature extraction, selection & reduction, model selection & tuning, normalization and post processing methodologies employed and compared, and the findings are reported.

## 1.0 Introduction

This study, examines the performance of the state-of-the-art supervised learning methods (i.e. classification and regression) in predicting some attributes of weather related tweets. The project is a online *Kaggle*[1] competition, sponsored by *CrowdFlower*. The competition involved weather related tweets with 24 classes' labeled using crowdsourcing. The task is to come up with data mining models and techniques to predict as accurately as possible for those 24 class variables for the test dataset.

Participants are provided with both *train* (class values are provided) and *test* (class values are available for the completion host but hidden from the competition participants) datasets. There were no restrictions on what techniques or tools the participants can use including any external data sources that the participants can tap into. The accuracy measure used for the evaluation is root mean squared error, RMSE. We built many models using Python and various machine learning libraries[2], *STATISTICA*[3] on the *train* set using various data mining techniques such as Multinomial Naïve Bayes, K Nearest Neighbors, Ridge Regression, Stochastic Gradient Descent (SGD) Classifier, boosted trees and random forest algorithms. We used cross validation to evaluate our models on the train data. Results from superior

[1] *Kaggle, http://www.kaggle.com.*
[2] *http://scikit-learn.org/stable/supervised_learning.html#supervised-learning*
[3] *STATISTICA by StatSoft, http://www.statsoft.com*

models were submitted to *Kaggle* web site. *Kaggle* automatically computes accuracy measure RMSE, and ranks the submission relative to those of other participants in the competition. Rank is computed based on the best submission made up until the current submission. Models based on the Ridge Regression have performed well.

There are about 250+ groups in the competition with 300 members. Our rank on the competition leader board ranged from 180s at the worst and 50s at the best. Finally, we ended up in the top 25 percent of the competition's leader board, ranked in the 60s.

We obtained a dataset labeled by crowd sourcing of about 80K weather related tweets which are used to predict continuous values of 24 classes for a test set size of about 42K. Each tweet is labeled by at least five people and weighted average of their grades are provided for each class. Continuous class values therefore are in the range of [0,1].

The format of the project gave us the motivation as these are:

- Interesting and challenging! Provided an opportunity to apply numerous things we learned in the class; no limits time permitting.
- Results are measurable against those of other experts in the data mining field, instantaneously.
- No restriction on the tools, techniques or additional data that one can use.

The goal of this project is to predict class variable values such that a metric, root mean square error (RMSE) on the test dataset is minimized, smaller the better.

Below is a brief summary of the competition.

- The competition started 9/27/13 and ended on 12/01/13.
- There is a cash prize of $500.
- Rank is computed on approximately 30% of the *test* instances for the public leader board. However, the private leader board is based on the 100% of the *test* instances which is only revealed after the competition ended.
- Each player/team is allowed a maximum of 5 submissions (predictions) per day.

## 2.0 Related Work

In recent years there has been tremendous amount of interest in analyzing microblog by the research community. One of the very related workshops to this problem is the recently organized first AAAI Conference on Human Computation and Crowdsourcing (HCOMP-2013) that was held November 6-9, 2013 in Palm Springs, California, USA. We are acknowledged of a shared task[1] in this workshop after the completion of the competition via a top contestant. Besides, there are many books, journals and articles published on opinion mining and sentiment analysis including Bing Liu's book that we used in the class.

[1] https://sites.google.com/site/crowdscale2013/shared-task/sentiment-analysis-judgment-data

Moreover, there are very nice conference articles related to microblog sentiment analysis published in ICWSM[1] and Making Sense of Microposts Workshop[2].

## 3.0 Solution

### Software Tools and Packages

- Python
  - Numpy, Scipy, Pandas
  - Scikit-learn, NLTK
- Java
  - Pre and Post Processing
  - Explored MULAN, an open-source Java library based on WEKA for multi-class multi-label predictions. Found to work only for categorical values so discarded the tool.
- *STATISTICA*
  - model building and predictions

We see this project as an opportunity for a survey study on available data mining tools and packages. Instead of diving into deeps in a very specific topic we utilized this class project for exploring the tools that are out there; because we believe that one cannot improve the state of the art without grasping it. Moreover, we believe that reinventing the wheels would be a waste of time and effort.

Python has become one of the most popular tools for data scientists. Based on our vague observations on the Kaggle forums, about 60% of contestants prefer it over other languages. Numpy, numerical analysis module provides fundamental data structures (arrays and matrices) and basic methods for linear algebra included in SciPy framework. SciPy (Python for scientific computing) library consists of modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. Matplotlib is a comprehensive 2D plotting tool that facilitates producing publication quality plots. Pandas provides high performance data structures. Lastly, IPython is a rich interactive shell that lets it's users process and test their ideas quickly. Moreover, IPython notebook works in web browsers and provides a wonderful way of documenting the computations. Our IPython notebook is also available in the appendix section. Below is a screenshot showing these core packages (from scipy.org).

[1] http://icwsm.org/2014/

[2] http://www.scc.lancs.ac.uk/microposts2014

**NumPy**
Base N-dimensional array package

**SciPy library**
Fundamental library for scientific computing

**Matplotlib**
Comprehensive 2D Plotting


**IPython**
Enhanced Interactive Console

**Sympy**
Symbolic mathematics

**pandas**
Data structures & analysis

We believe that Java is the most preferred programming language in computer science (at least in the academia). It has powerful object oriented framework and publicly available packages with a great community support. WEKA is a machine learning project developed with Java by University of Waikato, New Zealand. However, WEKA does not support multi-class multi-label predictions. That led us to look into another open source Java framework Mulan, a frame work based on WEKA and is also an open-source Java library for learning from multi-label datasets. But, soon we discovered that the Mulan is capable of predicting just categorical class labels but not the continuous values needed for this project. Hence, we discarded the tool. We then looked for COTS tools that can be licensed for academic purposes with nominal fee and has great features for statistical analysis and machine learning. We discovered that Statsoft's STATISTICA matched the criteria and hence was chosen for this project.

## Selected algorithms

- Regressors (continuous, clip [0,1])
  - Ridge Regressor (sklearn)
  - SGD (Stochastic Gradient Descent) Regressor (sklearn)
- Classifiers
  - MultinomialNB (cs780 hw2, sklearn)
  - KNN
- Ensemble methods, Regression (Using *STATISTICA***)
  - Boosted Trees
  - Random Forests

  *** Noticed the following limitations (may or may not be accurate as we have a very limited time to explore the tool): Required the input data to be in sparse format; No multi-class support for many models. Typical test/ train file with about top 4000 features was approx. 1GB. Typical execution on a 64 bit AMD 2.5 GHz Quad Core PC with 8GB of RAM took about an hour (Execution of each model consisted of 24 individual executions, one for each class, manual assembling of results into CSV file).*

### Classification Approach

Although class types were not nominal, we initially considered this competition as a classification problem rather than a regression to get started. However, regression turned out to be a better fit for us.

Our reasoning was based on the given information of grading. Simply, each tweet is classified by several humans and each grader chose only one class for sentiment and when categories while multiple classes could be selected for kind category. However, graders have some reliability weight. We are not given any information on particular graders instead we had weighted average of class values. So, we replicated each tweet in the training dataset as n_class times (for each category) and each time labeled it differently. This operation is illustrated in the figure below. We then used the real values as sample weights during the classification task. So, our dataset size has become 24 times bigger.

We fitted MultinomialNB and KNN classifiers on this new dataset along with sample_weight information. Later, predict_proba() functions of the models are used to predict the probabilities of the classes. How these probabilities are calculated by sklearn module in Python is out of scope of this paper.

| tweet | state | location | s1 | s2 | s3 | s4 | s5 |
|---|---|---|---|---|---|---|---|
| Jazz for a Rainy Afternoon: {link} | oklahoma | Oklahoma | 0 | 0 | 1 | 0 | 0 |
| | | | | | | | |
| | | | Class | Weight | | | |
| Jazz for a Rainy Afternoon: {link} | oklahoma | Oklahoma | 0 | 0 | | | |
| Jazz for a Rainy Afternoon: {link} | oklahoma | Oklahoma | 1 | 0 | | | |
| Jazz for a Rainy Afternoon: {link} | oklahoma | Oklahoma | 2 | 1 | | | |
| Jazz for a Rainy Afternoon: {link} | oklahoma | Oklahoma | 3 | 0 | | | |
| Jazz for a Rainy Afternoon: {link} | oklahoma | Oklahoma | 4 | 0 | | | |

**Regression Approach**

Since the values to predict are continuous, regressors are naturally the way to go. We have tried sklearn's Ridge Regression as well as SGDRegressor with various variables on several differently preprocessed datasets. Our best result is achieved by a Ridge Regressor.

We utilized the two of ensemble methods for regression available in STATISTICA, i.e., boosted trees and random forests. The goal of ensemble methods is to combine the predictions of several models built with a given learning algorithm in order to improve generalizability / robustness over a single model.

## 4.0 Experiments

### Data

Below are some salient points about the dataset.

- *Train* and *test* datasets used for the project were provided by *Kaggle*.
- *Train* dataset has 77,496 records and *test* dataset has 42,147 records.
- Each record consisted of id, tweet, state, location and 24 class variables.

NOTE: For test dataset the class variables were omitted so the task to predict them. Below is two sample records*:*

```
"1","Jazz for a Rainy Afternoon: {link}","oklahoma","Oklahoma", //id, tweet and location
"0","0","1","0","0", //sentiment
"0.8","0","0.2","0", //when
"0","0","0","0","0","0","0","0","0","1","0","0","0","0","0" //kind
"30","Today was windy AF, I don't wanna go meet up w/ @mention and get my sis from school
:/","arizona","Phoenix (Prescott)",
"0","0.613","0.387","0","0", //sentiment
"0.793","0","0","0.207", //when
"0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","1" //kind
```

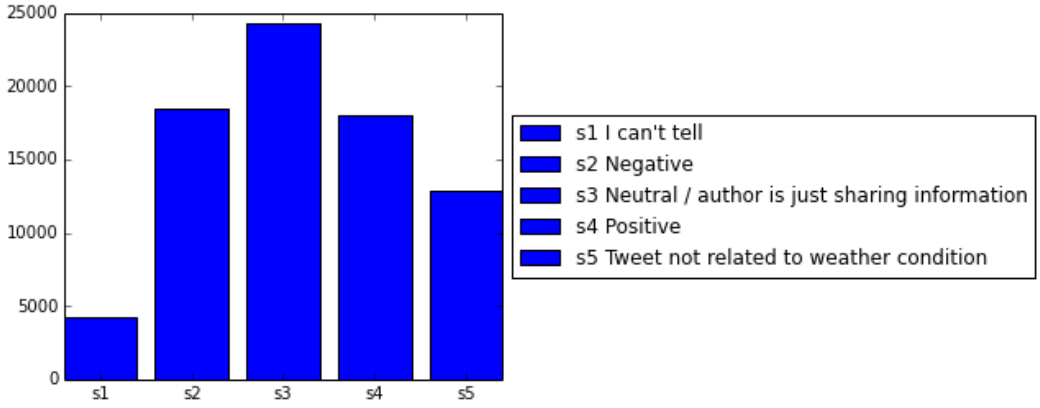
Table below shows the categories and the classes that belong to each of them.

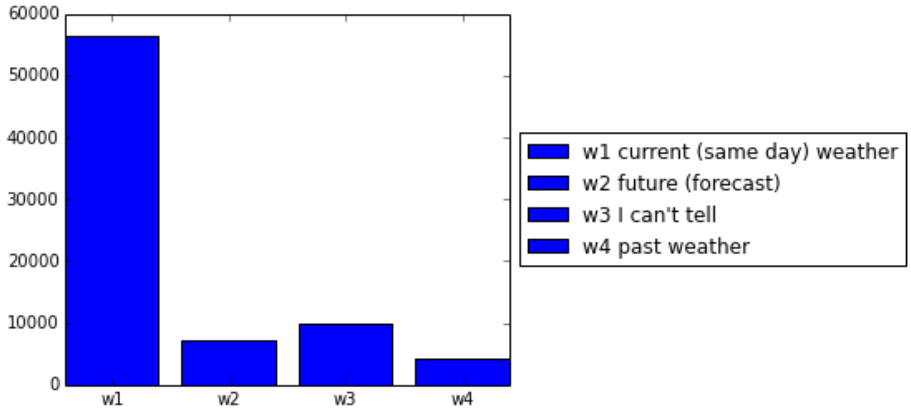
| Sentiment (5) | When (4) | Kind (15) |
|---|---|---|
| s1,"I can't tell" | w1,"current (same day) weather" | k1,"clouds" |
| s2,"Negative" | w2,"future (forecast)" | k2,"cold" |
| s3,"Neutral / author is just sharing information" | w3,"I can't tell" | k3,"dry" |
| s4,"Positive" | w4,"past weather" | k4,"hot" |
| s5,"Tweet not related to weather condition" | | k5,"humid" |
| | | k6,"hurricane" |
| | | k7,"I can't tell" |
| | | k8,"ice" |
| | | k9,"other" |
| | | k10,"rain" |
| | | k11,"snow" |
| | | k12,"storms" |
| | | k13,"sun" |
| | | k14,"tornado" |
| | | k15,"wind" |

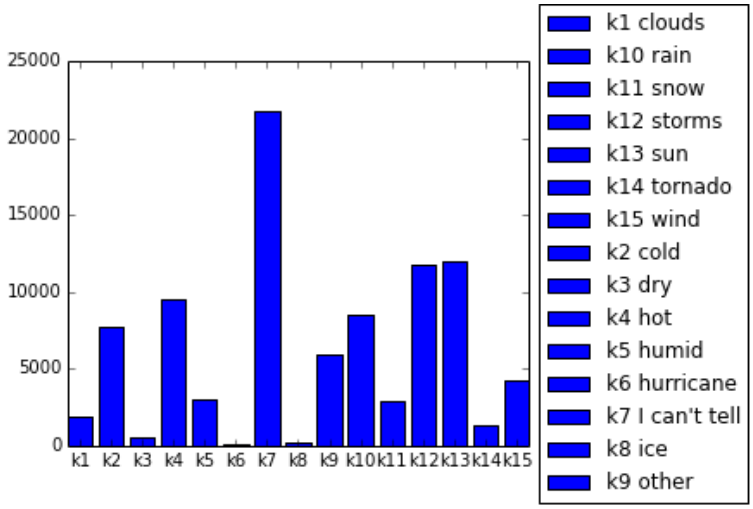
Here is how the class values are computed (per Kaggle website):

- Each tweet is labeled by multiple people (graders).
- Each grader is allowed to grade only one class from *Sentiment* with 1 and rest with zeros. So is the *When* category. However, they are allowed chose more than one class with a value of 1 for *Kind* category.
- Graders have reliability scores but we do not know them.
- The host of the competition normalized the values of classes such that the following condition is true.
  - Sum of the class variables in *Sentiment* category is 1.
  - Sum of the class variables in *When* category is 1.
  - Sum of the class variables in *Kind* category is not necessarily one.

Histograms we created by python's matplotlib library show the class distributions for different categories in the training dataset.




This is an imbalanced multi-class continuous value prediction problem.

## Experimental Setup

Various preprocessing, feature extraction, selection & reduction, model selection & tuning, normalization and post processing methodologies employed and compared, and the findings are reported in the following experimental results section.

We learned from 62356 tweets, i.e. 4/5 of the training set and predicted on the remaining 1/5. Then calculated the RMSE scores on this evaluation set. Throughout the document all RMSE scores reported are on this dataset unless stated otherwise. Since it would be infeasible to report effect of every parameter tuned in every step, results are presented in the following section is based on the predictions of Ridge Regression model (with alpha parameter set to 3.0) implemented in scikit-learn machine learning package in Python. Also, TFIDF vectorizer default parameter values used is as the following:

- use_idf = true
- ngram_range = (1,2)
- min_df = 5
- max_df = 0.5
- sublinear_tf = true
- smooth_idf = true
- max_features = 20,000

We used iPython interactive shell for experimental purposes where we were be able to observe the values in the variables and can investigate the matrices on the fly. Similar to MATLAB and Mathematica, interactive shell definitely boosts up the experimental setup process.

Benchmark baseline was all zeros y matrix which resulted in the RMSE score of ~ 0.31597. The goal of the project is to minimize the root mean square error:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}}$$ *where*

- $n$ is 24 times the total number of tweets
- $p_i$ is the predicted confidence rating for a given label
- $a_i$ is the actual confidence rating for a given label

## Experimental Results

### *Feature Extraction/Creation from raw tweets:*

The raw dataset supplied is not good enough to use directly with many data mining models. Hence, we needed to perform pre-processing data preparation. Many pre-processing techniques have been applied on both *train* and *test* data sets.

- Raw words, Porter/Lancaster/Snowball stemmers, wordnet lemmatizer (NLTK)
- Stop words removal (NLTK, scikit-learn)

- Unigrams, bigrams, together (scikit-learn)
- Emoticons (happy/sad)
- Stacking state and location information to tweets
- *STATISTICA*: for some selective runs, added additional features (Label_1 to Label_24) which are computed in case a synonym for any of the class label is found. Improved the RMSE by about 10%.

### Raw words, Porter/Lancaster/Snowball stemmers, wordnet lemmatizer (NLTK)

Both stemmers and lemmatizers try to return the root of a given word. However, stemmers stem a word based on some set of prefix and postfix rules while lemmatizers consider the vocabulary information and context, so they are much more slower.

All mentioned stemmers and lemmatizers are available in the NLTK module in Python. We could not see any significant difference between them and therefore decided to use the most popular one for this task, namely Porter Stemmer, for the stemming process.

Numbers of features and stored elements in the compressed sparse matrix in the training dataset where max_df = 0.5 (i.e. attributes appear more than half of the records are removed), and min_df = 5 (i.e. attributes appear less than five times are removed) are provided in the following table (after stop words removed).

| **Without stemming (77946 tweets)** | | **With stemming (77946 tweets)** | |
|---|---|---|---|
| **# of unigrams & bigrams** | **# of elements** | **# of unigrams & bigrams** | **# of elements** |
| 27,348 | 1,274,752 | 27,404 | 1,288,126 |

Table 1 Number of attributes and their non-zero values : with and without stemming

### Stop words removal (NLTK, scikit-learn)

Stop words is a set of words that does not provide any information for solving a problem. Prepositions, articles, and alike usually appear in this list. This list might change according to the context of the problem. Besides, ready to use stop words are provided by software packages for common use. We also employed both NLTK's and scikit-learn's 'english' stop word removal operations to our dataset.

Removing stop words in computational linguistics is a common method for reducing the size of the dataset. Moreover, one can focus on the important words by removing them because they appear in the most common list and tracing the related information becomes more difficult during analysis.

| **Without stemming (77946 tweets no stop words)** | | **With stemming (77946 tweets no stop words)** | |
|---|---|---|---|
| **stop words not removed** | | **stop words not removed** | |
| **# of unigrams & bigrams** | **# of elements** | **# of unigrams & bigrams** | **# of elements** |
| 40,097 | 2,001,704 | 40,365 | 2,017,916 |

Table 2 Number of attributes and their non-zero values : with and without stop words

### Unigrams, bigrams, together (scikit-learn)

We measure our model with different features. Initially started with a bag of words, i.e. unigrams only. Later decided to use bigrams as well because it can capture some other information, which is impossible to catch with bag of words, such as negations. "not happy" and "happy" are definitely two important phrases that should be differentiated. Below we report our ngram combination RMSE results:

| Without stemming | | | |
|---|---|---|---|
| **Unigram only** | **Unigram + bigram** | **[1,2,3] grams** | **bigram only** |
| 0.1557 | 0.1511 | 0.1515 | 0.1725 |

Table 3 contiguous word ngrams RMSE scores

**Emoticons (happy/sad)**

Most common 20 emoticon list is obtained from datagenetics.com[1] website where they collected ~100M tweets and extracted 2242 unique smileys. It is reported that top 20 emoticons account for the 90% of all occurrences. A regular expression substituted emoticons either with "happy" or "sad" word.

| With stemming | |
|---|---|
| **RMSE with emoticon parsing** | **RMSE without emoticon parsing** |
| 0.1536 | 0.1539 |

Table 4 RMSE scores with and without emoticon parsing

**Stacking state and location information to tweets**

"id,tweet,state,location" is the format of dataset. At the very beginning we only tried to learn from tweets assuming that the location is independent of the classes to predict. We were wrong as the RMSE scores below indicate that our model predicted better when we stacked state and location information to the tweet texts.

| Without stemming | | |
|---|---|---|
| **tweet (text) only** | **Text + state** | **Text + state + location** |
| 0.1530 | 0.1512 | 0.1511 |

Table 5 RMSE scores with and without state and location information

*Feature reduction:*

- Selecting based on TFIDF importance
- LSA (truncated SVD, work with sparse matrix, on sample vec)
- PCA (dense only, on covariance matrix)
- SelectKBest(chi2, class/feature dependency)
- *STATISTICA*: Utilized the best feature selection option for each individual classes. Used top 100 of such features for each class.

**Selecting based on TFIDF importance**

Instead of using all features (i.e. all unigram and bigram tfidf fitted and transformed data), we decided to remove less informative features. Number of features was provided in the previous section, and we decided to measure our model prediction success with 20K feature limit, 10K feature limit, and 1000 features limit. Please recall that number of features in the initial dataset was about 30K as stated in the previous sectoin. Resulting table is provided below:

| Without stemming | | With stemming | |
|---|---|---|---|
| **RMSE without limit** | **20K feature limit** | **RMSE without limit** | **RMSE 20K with features** |
| 0.1510 | 0.1511 | 0.1536 | 0.1536 |

[1] Study is available at <http://datagenetics.com/blog/october52012/>

| 10K limit | 1000 limit | 10K limit | 1000 limit |
|---|---|---|---|
| 0.1518 | 0.1627 | 0.1544 | 0.1649 |

Table 6 RMSE scores with and without feature limitations

**Truncated Singular Value Decomposition (LSA)**

LSA performs linear dimensionality reduction by means of truncated SVD, and directly works on sample vector, so its implementation in sklearn module accepts sparse matrix, which is extremely faster than the dense matrix representation. We reduced 20K features to 300 and 1K however both gave an RMSE score of ~0.18 in CV that deteriorated the previous scores quite a bit. Here is the related code snippet:

```
>> model = linear_model.Ridge (alpha = 3.0, normalize = True)
>> pred,y_true = cv_loop(train, t2, model,is_LSA=True)
Train error: 0.181060399966
```

**Principal Component Analysis (PCA)**

PCA, unlike LSA works on covariance matrix and can only work with dense representation therefore much more slower than LSA in sklearn module. It also resulted in segmentation fault in our system.

**SelectKBest of Chi-squared**

Computed $\chi^2$ (chi-squared) statistic for each class/feature combination and selected the most k informative ones. The problem with this reducer was that it worked with binary class values and so resulted in a serious distortion in our case: gave 0.2473 RMSE score. So we decided not to employ this reduction.

We used several algorithms described in the Solution Section to predict class values on the preprocessed data where features are extracted and reduced by the techniques mentioned in this section. Later, cross validation method is used to evaluate the goodness of the algorithm. If the cross validation resulted in a better RMSE, then we decided to upload such results on to Kaggle website.

*Normalization:*

Since, each class variable is predicted independently, they did not always sum up to 1 for Sentiment and When categories. Therefore, we normalized them to result in a value of one. Moreover, regressors returned negative values which is unacceptable for a prediction probability. So, we clipped their prediction to [0,1]. By clipping we mean that any value less than zero becomes zero, and any value larger than one becomes one.

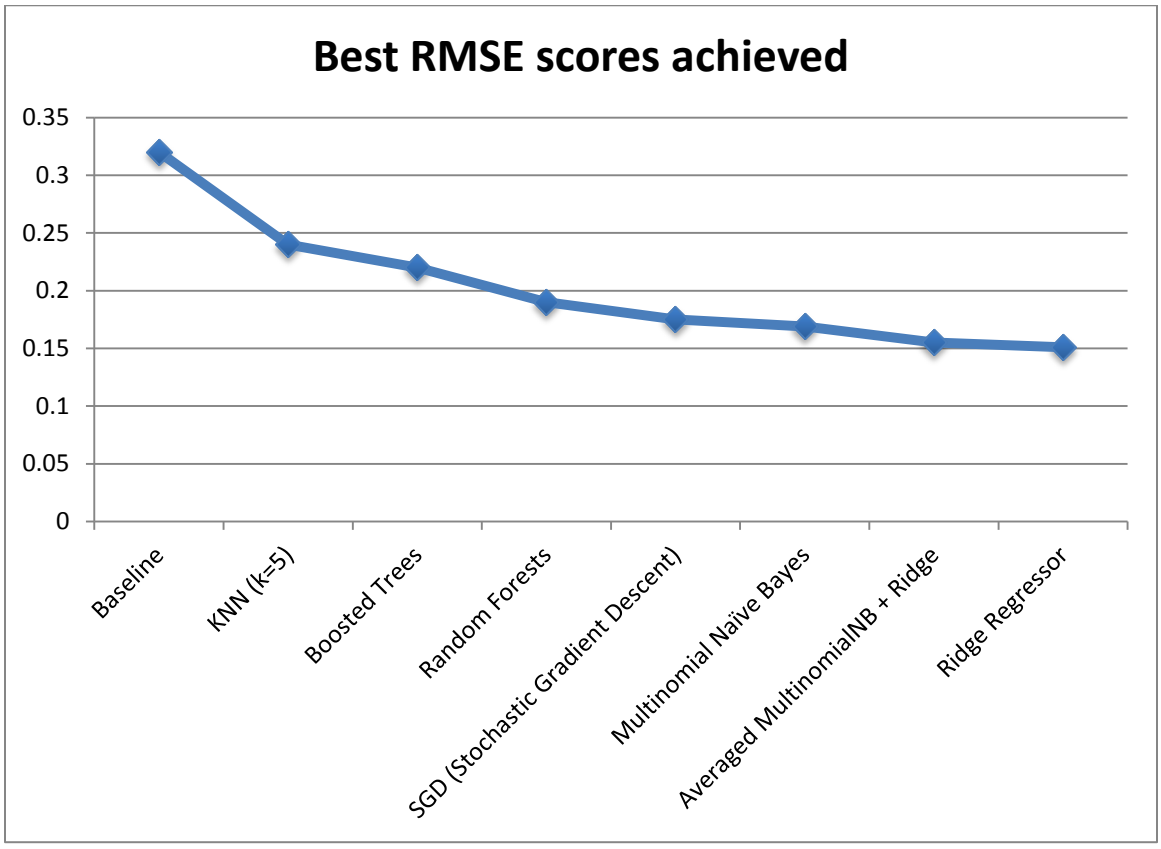
**Models**

We had introduced our solution approaches for both classification and regression in the solution section before. Baseline is all zeros prediction set by contest organizer Kaggle and gives an RMSE score of 0.32. The models achieve different scores for different set of features and the way their parameters tuned. Here we are just reporting the best results came out of these models. Details of how the dataset is preprocessed and how the parameters selected and what was their effect on the models are provided in the previous section.

| Classifier/Regressor | Best RMSE scores achieved |
|---|---|

| | |
|---|---|
| Baseline | 0.32 |
| KNN[1] (k=5) | 0.24 |
| Boosted Trees | 0.22 |
| Random Forests | 0.19 |
| SGD (Stochastic Gradient Descent) | 0.175 |
| Multinomial Naïve Bayes | 0.169 |
| Averaged MultinomialNB + Ridge | 0.155 |
| Ridge Regressor | 0.151 |

Table 7 Best RMSE scores achieved for each model




## Summary and Conclusion

Huge part of the effort was pre-processing and tuning various different models to see which model gives the best RMSE. We implemented in Python and employed its popular data mining libraries.

We also used custom Java libraries and STATISTICA

[1] At the time of writing this report, we could not verify KNN model. We had to kill our process because of the submission deadline. As far as we remember, the results were not promising and were in these lines.

- Developed custom Java programs for pre and post processing.
- Normalized predicted class labels STATISTICA to sum to a desired value, e.g., 1.0 for sentiment and when labels.
- Models based on Ridge Regressor have performed extremely well.

Uploaded many predictions onto Kaggle website, about 35 (1% of all participant uploads).

Although ridge regression outperformed in this multiclass continuous value problem in our experiments, highest ranked contestant was able to use classifiers in a better way and was able to get higher RMSE scores. First ranked contestant has RMSE score of 0.14314.

What we learned from top contestants (thanks to the Kaggle online discussion forum):

- Simple stacking: feed the output of a first level model to use it as features to the 2nd level model
- Add more features: POS tagging, sentiment dictionary, etc. (1st ranked fellow had ~1.9M features)

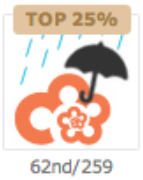
Competition started about two months ago. Some teams had 100+ submissions.

**Started:** 2:43 pm, Friday 27 September 2013 UTC
**Ended:** 11:59 pm, Sunday 1 December 2013 UTC(65 total days)

Final Numbers from Kaggle website:

259 teams
300 players
3604 entries

We entered just two weeks ago and are glad to report that we made it to the top 25% of the leader board by the time the competition finished on Dec. 1st 2013.



62nd/259

## *References*


- *NLTK (Natural Language Toolkit), http://nltk.org/*
- *Scikit-learn Machine Learning in Python, http://scikit-learn.org/stable/*
- *Pandas Python Data Analysis Library, http://pandas.pydata.org/*
- *StatSoft's STATISTICA 12 (64-bit edition) Academic License, http://www.statsoft.com/*

## *Contributions of team members*

- *We discussed various techniques for preprocessing the raw data: stemming, stop word removal, etc. Continuously discussed what is working and what is not and what the next steps are.*
- *Talha used Python and various machine learning libraries available. Venkat used Java, explored MULAN, and finally used STATISTICA.*
- *Both the members contributed equally in the preparation of PowerPoint Presentation and this Project Report.*
- *Special thanks to some of the other members of the competition on the Kaggle Discussion Forum. They shared a bootstrap Python code and some of their valuable tips and techniques.*