# Kaggle See Click Fix Model Description

**BY:** Miroslaw Horbal & Bryan Gregory
**LOCATION**: Waterloo, Ont, Canada & Dallas, TX
**CONTACT** : miroslaw@gmail.com & bryan.gregory1@gmail.com
**CONTEST**: See Click Predict Fix (http://www.kaggle.com/c/see-click-predict-fix)

## Summary

Our final model consisted of a segmented weighted average of our individually developed models which consisted of a linear regression model (Miroslaw's), and a segmented ensemble of tree-based gradient boosting regression models and linear regression models (Bryan's).

We identified 5 distinct segments in the data and used these segments for the segmented ensemble base model (Bryan's model), in applying segmented scaling used in both base models, and in segmented averaging of the final ensemble. The segments consisted of remote_api sourced issues and the cities (Chicago, Oakland, New Haven, and Richmond).

The primary features in the linear model (Miroslaw's) were a TFIDF weighted bag of words, natural log-transformed description length (word counts), and one hot encoded categorical features.

The segmented ensemble base model consisted of tree-based models and linear regressions models that were segmented based on source and city and each of those individual models relied on one hot encoded categorical features and population-based features mined from US census data.

Both the linear model and tree-based models included scaling parameters that scaled down their final predictions based on the target variable and the data segments. Both models also took advantage of zip codes and neighborhood names mined from the free and open source MapQuest nominatim database using the longitude and latitude for reverse geocoding.

Our ensemble combined the output of our individual models by applying segment based weights to each base model. The weights were derived by applying linear regression to optimize cross-validation (CV) scores, then the weights were manually adjusted using heuristics and leaderboard feedback, with the underlying goal being that our linear regression weights were not overfitting the CV data set.

## Feature Selection / Extraction

Miroslaw's linear model's features were selected using a manual greedy forward selection process using the final 20% of the training data for cross validation. Preprocessing of the data consisted of replacing any empty description, source, and tag_type fields with a '__missing__' string. The features for the linear model are:

- A TFIDF bag of words on summaries and descriptions using unigrams, bigrams, and trigrams
- A binary indicator variable if the issue was created on a weekend
- A numerical feature computing log( # of words in description + 1 )
- Categorical one hot encoded features for:
-- City Name
-- Time of Day split into 4h segments
-- Neighborhood Name
-- Zipcode
-- Source
-- Tag Type
-- City + Time of Day
-- City + Source
-- City + Tag Type
-- Source + Tag Type
-- Zipcode + Source

Bryan's segmented ensemble model was developed after identifying that there was a lot of interaction between the features within 5 distinct segments in the data. The goal was to account for that feature interaction by training base models on each segment, then combining the output for a total submission. The features for Bryan's segmented ensemble varied by segment because the segments varied in how they responded to varying features. The exact features used for each can be found in the MODELS.json file stored in Bryan's code base.
The total list of features used during development was nearly identical to those used in Miroslaw's linear model, listed above. The only notable exception being that a few of the segments responded more favorably to linear description length counts, so that additional feature was utilized in a few base models.

## **Modeling Techniques and Training**

Miroslaw's model was focused around the rich text data found in the summary and description fields. However the challenge in mining text data is the huge feature dimensions that are derived, particularly when using n-gram combinations. This greatly reduced the number of potential models from which to choose, limiting to mostly linear models. Ridge regression proved to be most in effective in both cross-validation scores and leaderboard feedback.

For Bryan's model, the features, learning algorithm, and parameters used for each base segment model were chosen through a mix of cross-validation scores and leaderboard feedback. The segments varied in how they responded to various learning algorithms, however in general Gradient Boosted Machines and Stochastic Gradient Descent Regressors performed quite well. GBM's trained fairly quickly in general thanks to the segmentation of the data creating a more manageable data set for each model. The segmentation approach had the added advantage in that it allowed me to explore more than just linear models thanks to the reduced dimensionality.

Together our individually developed models performed amazingly well when combined into an averaged ensemble. They then performed even better when we moved towards segment based weights. Finally our last big improvement came in using linear regression to derive the ideal weights based on cross-validation scores. We combined those segment based weights derived from linear regression with basic heuristics and leaderboard feedback to determine the weights for our final winning model.

## Code Description

We both chose the popular python stack of SKLearn/PANDAS/NumPy. We both have backgrounds in Python and the language allows for rapid development of models, and the SKLearn library is a very robust library of tools useful in machine learning. All libraries used are free and open-source.

## Dependencies

Dependencies for both models are: python > 2.6, scikit-learn, ipython, scipy, numpy, and pandas. All libraries used are free and open-source.

## How to Generate the Solution

For Miroslaw's model on ubuntu 12.04 LTS:

- Install dependencies:
  - sudo apt-get install ipython python-scipy python-sklearn python-pandas
- Download code from github:
  - git clone https://github.com/beegieb/kaggle_see_click_fix.git
- Extract official data, geodata, and bryans preprocessed data:
  - cd kaggle_see_click_fix
  - unzip -d data data/train.zip
  - unzip -d data data/test.zip
  - unzip -d data data/geodata.zip
  - unzip -d data data/bryans_data.zip
- Make a cache and submission directory:
  - mkdir cache, submissions
- Run test.py with the final model "ridge_39":
  - >>> python test.py ridge_39
- The final model's predictions will be saved to "submissions/ridge_39.csv"

For Bryan's model:

- Install dependencies:
  - sudo apt-get install numpy python-scipy python-sklearn python-pandas
- Download code from github:
  - git clone https://github.com/theusual/kaggle-seeclickfix-model.git
- Data is already extracted and found in the /Data/ directory. Other subdirectories included are: /Cache/, /Logs/, and /Output/.
- Everything has been pre-configured to run correctly. Run main.py with the default settings in SETTINGS.json:
  - >>> python main.py
- The predictions will be saved to "Output/bryan_test_predictions.csv"
  - If existing predictions already exist, they will be renamed to append their creation date and time to the end, and then moved to "Output/Archive"

For the final ensemble:

- Install dependencies:
  - sudo apt-get install numpy python-pandas
- Download code from github:
  - git clone https://github.com/theusual/kaggle-seeclickfix-ensemble.git
- The individual base models are found in the subfolders /Bryan/ and /Miroslaw/. They already have existing submission outputs that can be used by the ensemble code to create a final submission, OR the individual base models can first be run to re-create their submission outputs prior to running th ensemble code base.
- Everything has been pre-configured to run correctly. Run main.py with the default settings in SETTINGS.json:
  - >>> python main.py

## Additional Comments and Observations

Miroslaw:

Both Bryan and I were very surprised at how effective applying scaling to our models final predictions was. Early on in the competition applying a scaling factor of 0.81 for votes and 0.89 on comments was able to improve the performance of my model's private leaderboard score by 0.0065. This was further improved with more advanced scaling techniques such as applying an exponentially decaying scale based on the number of months a test issue was from April 31 2013. My final model's scales were based on city name and if the source was labeled as remote_api_created.

I spent a lot of time focused on trying to train a novel implementation of a Restricted Boltzmann Machine that can quickly train on high dimensional data for dimensionality reduction. While I was successful in developing a working implementation of a SparseRBM and showing experimentally that it was superior to a linear model on purely binary features, it was unable to match the performance of a linear model trained on TFIDF weighted bag of words. It seems that a TFIDF weighted bag of words provides superior information as a feature than a binary bag of words that even deep-learning techniques could not match. I suspect that a GPU trained Deep Neural Network using the features I used in my linear model could be superior to my linear model but unfortunately do not have the computational resources to experiment with that hypothesis at this time.

I also tried to train a gradient boosting regression model on the dataset generated by the SparseRBM but ran into difficulty with matching leaderboard scores to my cross validation scores. I suspect the reason why Gradient Boosting failed is a result of my inexperience and lack of understanding good training techniques for that class of model.

<u>Bryan:</u>

Identifying the distinct segments in the data proved to be very effective for us in many ways as they were used in my base model, in our base models' scaling, and in our final averaged ensemble.

For me, the largest simple improvement came from segment based scaling. By performing a simple grid search using cross-validation feedback, one could derive the optimal scalars for the 5 data segments and the 3 targets very quickly, and this gave a huge improvement in accuracy to the model. We believe this effect exists because of the need to compensate for the temporal effect of the data (using historical performance to dictate a changing future target) and because of the nuances of the accuracy metric chosen (RMSLE).

I also found that Gradient Boosted Machines performed very well on this data set, both in the segmented ensemble and earlier solo models I tested. That is likely due to the way in which GBM's tend to emphasize a few strong features and marginalize less important features, and this dataset certainly seemed to have the strongest signal in just a few of the main features.

## Simple Features and Methods

<u>Miroslaw:</u>

I found the single best improvement to my model came from using the final three months of training data. Overall I improved my score on the private leaderboard by 0.03691 after removing any training data prior to Feb 2013. Similar gains were seen in CV scores.