What is the CrowdFlower competition about?

The goal of the CrowdFlower competition was to come up with a machine learning algorithm that can automatically evaluate the quality of the search engine of an e-commerce site. Given a query (e.g. ‘tennis shoes’) and an ensuing result (‘adidas running shoes’), the goal is to score the result on relevance, from 1 (least relevant) to 4 (most relevant). To train the algorithm, teams had access to a set of 10,000 queries/result pairs that were manually labeled by CrowdFlower using the aforementioned classes. More details about the competition are available here.

Tell us about the makeup of your team

Our team ‘Quartet’ was formed by Maher Harb, Roman Khomenko, Sergio Gamez, and Alejandro Simkievich. This was a very multi-faceted and multi-cultural team indeed, with folks ranging from physicists to computer scientists to business majors and nationalities from Canada, Ukraine, Spain, and Brazil. While diverse, we are guilty of being an all-male team and yet we hope we can fix that in the next competition (ladies, join our team next time!).

What was your overall approach to the challenge?

Our approach consisted of implementing the following steps:

1. Cleaning the original datasets (e.g. removing stopwords, removing special characters, converting characters to lowercase, stemming, etc.) and applying the TF-IDF (term frequency, inverse document frequency) algorithm to a concatenation of the query, product title, and product description.

2. Applying dimensionality reduction to the resulting datasets to work with roughly 250 instead of thousands of features.

3. Augmenting the resulting datasets with custom engineered features.

4. Building three different model families, including support vector machines (SVM), artificial neural networks (ANN), and gradient boosting machine (GBM).

5. Rounding the regression predictions to integers using specially developed thresholds that optimize the kappa score.

6. Ensembling the individual models and building second-layer models. *The winning model was an ensemble of a single-layer ANN, a single-layer SVM, and a second-layer ANN (with features derived from over 20 single-layer models)*.

What processing methods did you use?

The original dataset was raw text. A number of tasks, discussed extensively at the Kaggle forum and that most competing teams carried out, were performed to convert the raw data to one on which machine learning algorithms can be trained. First, the datasets were cleaned by applying steps that included:

- Removing html tags and other non-text elements. These elements were present since, presumably, the datasets were generated programmatically by parsing the web.

- Removing stop-words such as common articles, prepositions, etc. (e.g. 'the', 'a', 'with','on').

- Using word-stemmers so that similar terms such as 'root' and 'roots', or 'game' and 'gaming' are converted to the same base word.

Basic features were generated by applying the TF-IDF operation on the resulting text. TF-IDF is an algorithm that assigns a value between 0 and 1 to every word in a given document, which in turn is part of an existing corpus. The more frequent a term is in the document and less frequent in the corpus, the higher the value for such term. For more details on TF-IDF, please check this wikipedia article.

Because the TF-IDF creates a feature for every token in the corpus, we used dimensionality reduction to end up with a more workable dataset of 225-250 features, depending on the model. The particular algorithm we used for dimensionality reduction is truncated singular value decomposition (SVD). More information on SVD is available here.

Where there any innovative aspects to your approach?

As it turned out, the quality of the models could be enhanced significantly by creating ad-hoc features that allow all machine learning models alike to make better decisions. Therefore, feature engineering was an important factor in this competition. The following set of features were added to the training and test datasets:

N-gram similarity features - A total of 14 features were added through the evaluation of similarity between unigrams, bigrams, and trigrams extracted from the query and the product title & description strings. Some of those features are boolean (e.g. 1 if unigram or bigram is found in search string, 0 if it is not) while others are similarity scores between the n-gram and the search string. To fully understand these features, we recommend taking a look at the R code used to generate the features. However, as a representative example of the general idea, the following demonstrates how one of the 14 features was created:

- **Query**: "*led christmas lights*"
- **Title**: "*Set of 10 Battery Operated Multi LED Train Christmas Lights - Clear Wire*"

6 possible n-grams can be generated from the query string. Those are ["*led christmas lights*", "*led christmas*", "*christmas lights*", "*led*", "*christmas*", "*lights*"]. Searching the product title for exact match of the query n-grams yields [0, 0, 1, 1, 1, 1]. The sum of the result (4) normalized by the maximum number of possible matches (6) becomes a feature (0.67).

Query-product name similarity features - While checking the cross-validation results, we noticed that our solution performed particularly bad whenever the main noun in the query was present in the product title, but it was not the actual product (i.e. it was an adjective in the title rather than the main noun). An example would be: query = "*night gown*" and product title = "*night gown necklace*". Thus, we developed an algorithm to extract the main noun from the product title and performed a similarity measure between the query and the extracted noun. This was accomplished by implementing a set of rule-based regular expressions that were carefully crafted to remove all unnecessary descriptions, prefixes, suffixes, etc. For example any phrase following the words "for", "by", and "with" was removed, sizes & colors were removed,and so on. Finally, the last two words in the cleaned title were regarded as the main product. The following is a good example of what is achieved through this algorithm:

- **Query:** "*reusable straws"*
- **Title:** "*Itzy Ritzy Snack Happens Reusable Snack Bag with Thermos Straw Bottle, Jungle"*
- **Cleaned title:** "*Itzy Ritzy Snack Happens Reusable Snack Bag"*
- **Product name:** "*Snack bag"*

For additional details refer to the R code used for generating this feature.

Alternative query similarity features - For each of the 261 unique queries, we created a corpus by combining all titles associated with that query, extracted the top trigram, and used it as an alternative query. This basically allowed us to potentially double our engineered features. The idea behind this approach was to capture the correct product implicated when the query did not have similarity with the product title. This is especially an issue when the search query is a brand name. As an example, figure 1 below is a word cloud visualization of the trigrams in the title corpus associated with query = '*samsonite*'. The top trigram is *'piece luggage set*'. Because class 4 is the most common class (accounting for ~60% of the all labels), a reasonable assumption was made that the intended product is indeed a '*piece luggage set*'. Thus, we considered '*piece luggage set*' as an alternative query and subsequently generated similarity features between it and the product title/description. For additional details refer to the [R code] used for generating this feature.

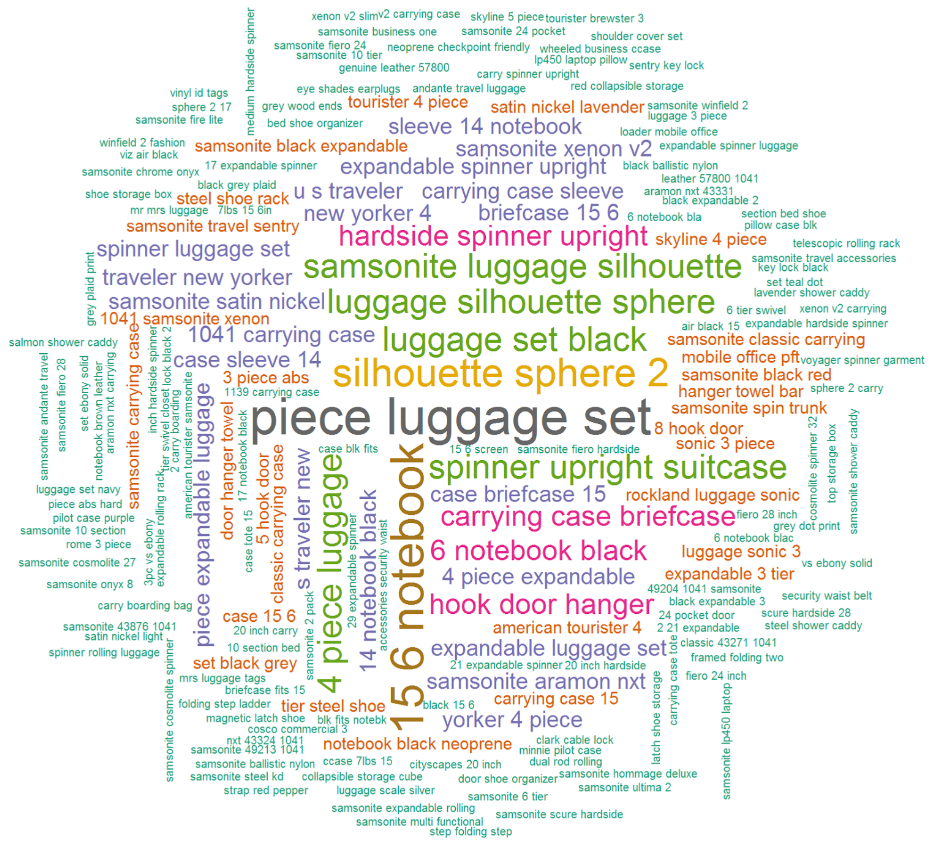

Figure 1. Word cloud of trigrams extracted from the titles corpus corresponding to query 'samsonite'

Intra-title similarity features - The idea behind this feature is to use the group of titles with the same relevance and within the same query as reference point and measure similarity of all individual titles within that query to that reference point. So, for instance, if a certain query has 40 labeled titles (10 per relevance class), then various similarity measures between each title (both labeled and unlabeled) and the 4 groups are constructed. In practice, this is carried out by measuring similarity between the title and each title within the group, then aggregating the result. Three different similarity measures were used: cosine similarity, count of same words, count of same words after extracting the product name, and two different aggregation functions were applied (mean and max), yielding a total of 4✕3✕2= 24 features. For additional details refer to the python code used for generating this feature.

Antonym feature - Noticing that some queries performed particularly bad when compared to hand-labeled predictions in some particular cases, we developed some additional rules to address this issue. One such case was when an antonym of a noun in the query was in the product title or description (e.g. query: “*men shoes*”, product title: “*beautiful women shoes*”). To solve for that, we created a boolean variable indicating whether an antonym is present (1) or not (0). This was done only for the most common antonyms.

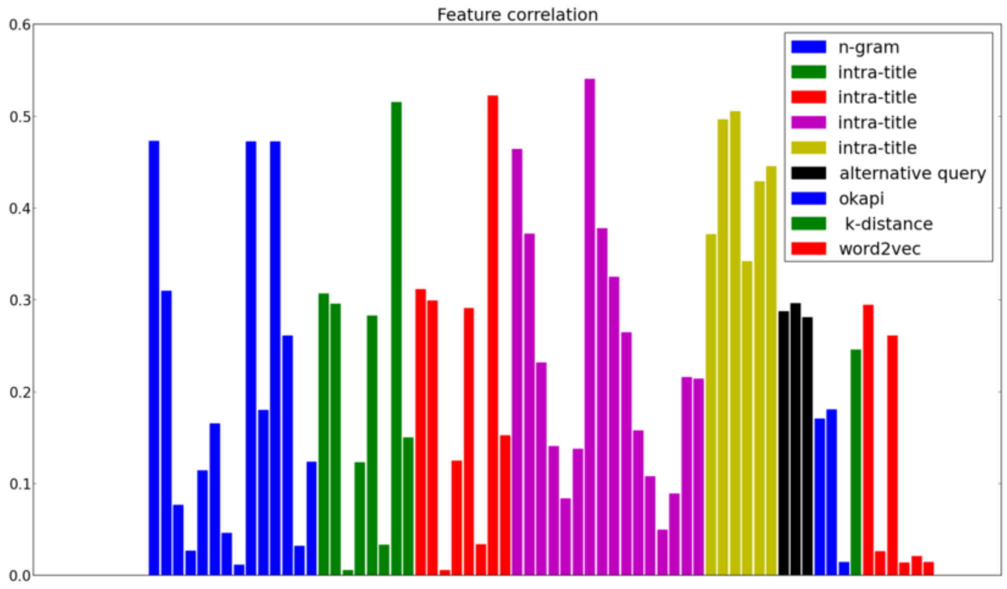

Figure 2. Bar plot of the correlation between selected engineered features and label

What supervised learning methods did you use?

We used a variety of models that included support vector machines, artificial neural networks, gradient boosted regressors, k-nearest neighbor classifiers, and random forest classifiers. For some of the models, we created both classifications and regression versions (e.g. svm classifiers and svm regressors). We also used alternative implementations of the same model (e.g. using both XGBOOST package and H2O to build gradient boosted regressors). As a result we had a wealth of different models to be used in ensembling and building of second-layer models (in specific, the second-layer ANN was based on features from over 20 different models).

Implementation wise, SVM was built using the excellent machine learning library of scikit learn (hyperlink). The artificial neural networks were built using keras (which in turn outsources some tasks to theano) on python. Whereas GBM models were built using the XGBOOST and H2O R packages.

How did you round the regression outputs to yield classes?

By running experiments on the cross validation folds, we determined the optimal rounding thresholds for rounding predictions obtained through regression. In the end, we found that the following thresholds produced the best kappa score: 1.9, 2.7, 3.5. (i.e. if prediction is lower than 1.9 then label 1 is assigned, if bigger than 1.9 but lower than 2.7, then label 2 is assigned, and so on).

Describe your winning ensemble

Our final model (depicted in the diagram below) is a 2-step ensemble, which can be described as follows:

Step 1 - Ensemble of first-layer ANN (30% weight) and second-layer ANN (70% weight). The averaging is applied on the unrounded predictions and the optimal rounding thresholds were applied after averaging.

Step 2 - Ensemble of the predictions obtained from step 1 and SVM classifier, according to the following expression: floor(0.8*step1 + 0.35*SVM). The expression looks rather arbitrary, but when carefully examining the possible outcomes of combining the two models according to this formula, we noticed that this simply applies a slight correction to predictions of step 1. Most notably, when step 1 predicts class 2 and the SVM predicts class 1, class 1 is chosen. Our guess for why this operation boosts the overall score is due to the SVM being better at classifying class1 than the step1 ensemble.

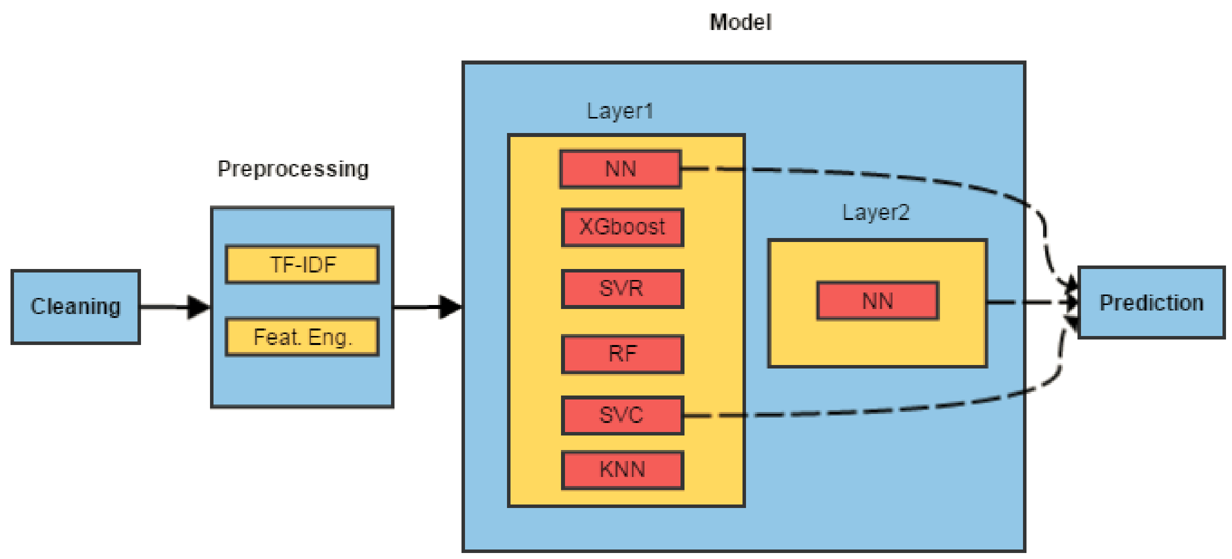

Figure 3. Block diagram depicting the overall solution

What have you taken away from this competition?

As can be seen from this blog, many different ideas were applied. Those ideas can be grouped into a number of initiatives such as feature engineering, testing different models, testing different machine learning tasks such as regression vs. classification, and trying different ensembling and rounding strategies. Not only does it take a lot of time to test so many different ideas, but a lot of ingenuity needs to be put into idea generation. While different team members contributed differently to different initiatives, everyone was actively looking for ways to improve the overall solution and the competition score. This leads us to one issue that is often discussed in machine learning and management overall: the power of teams and teamwork.

In machine learning, it is well known that the product of model ensembling is usually better than any individual model. The simple reason is that every model suffers from errors due to variance, bias, and irreducible errors. Ensembling models allows for correcting of those errors, at least in part, and as long as the models are independent. Subsequently, a team of data scientists with different approaches to the same problem can create a diversity of models more than an individual, and thus can ensemble more models. So, all things being equal, the product of those ensembles will be better the larger the number of data scientists engaged in creating models.

But that is not all. Idea generation is a tiresome process, and at different points in time, different team members can suffer from burnout. When that happened among us, someone was always willing and ready to take the lead. Some of us are 100%

convinced that in a real-life setting, where companies compete aggressively for their customers' dollars, individuals cannot compete with teams, no matter how good the individual. This is a lesson to all of us. While exceptional data scientists do win competitions by themselves - and this particular competition is one of those - more often than not the top 10 in Kaggle competitions is comprised of teams. When you move to real life, where serious money is involved, you can expect individuals to team up to stand a higher chance to win in the marketplace. Do not try to climb the Everest all by yourself - doing it alongside a team is more fun and will probably win you more money in the end.

About team 'Quartet':

Maher Harb - Physicist and data scientist. Starting sept 2015 will take on a new position as Assistant Professor at the Dept. of Physics at Drexel University (Philadelphia).

Roman Khomenko - Senior software developer and security researcher from Ukraine.

Sergio Gámez - Computer vision researcher from Spain.

Alejandro Simkievich - Technology entrepreneur and CEO at Statec, a machine learning consulting company in Sao Paulo, Brazil