Search Relevance Product Case

Alberto Gabriel Buftea

Stages

1. Query understanding

- Text Pre-Processing
- Query Spelling
- Query Classification
- Query Expansion
- Query Tagging

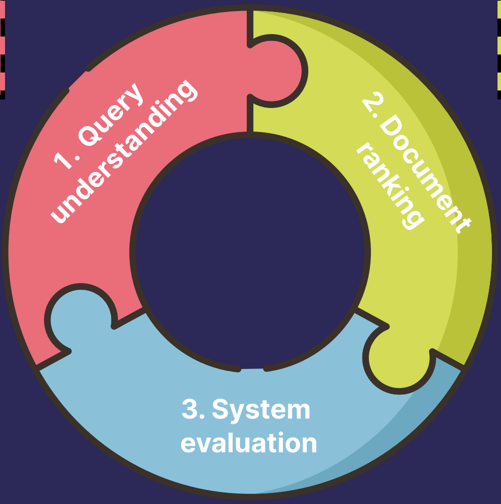
2. Document ranking

Tactical Solution

- Boosting Ranking Strategy

Strategic Solution

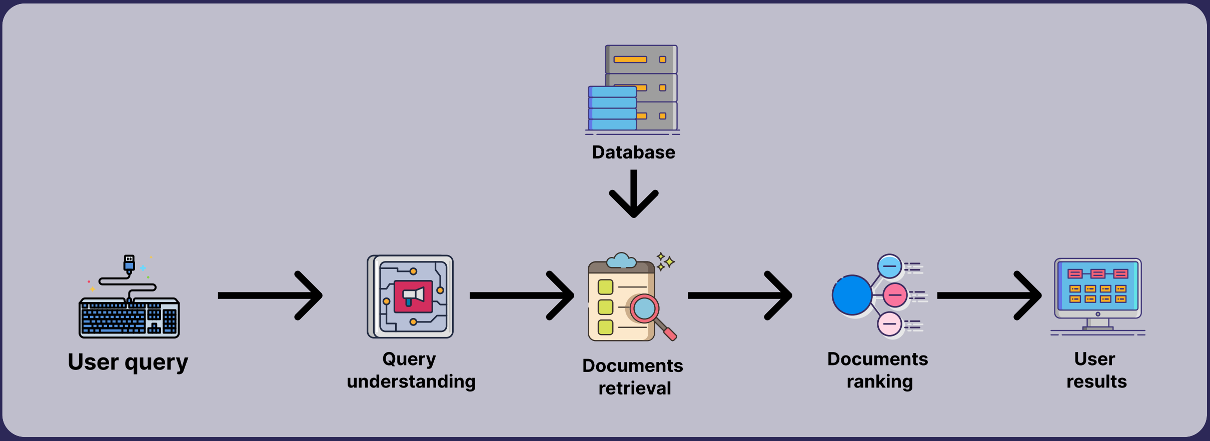
- Learning To Rank - List Wise Approach



3. System evaluation

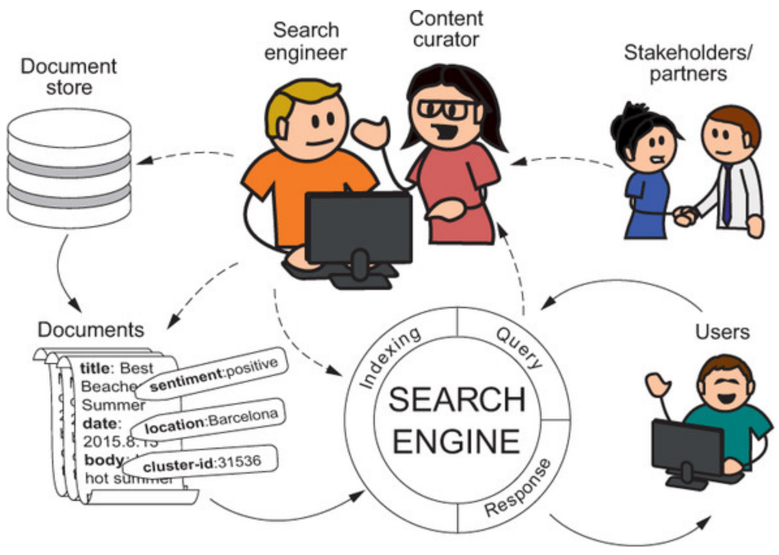
- Effectiveness
- Efficiency
- Reliability

System Design



Following Discussion

Further system information can be reviewed in the article below.



ML Product Case: Search Relevance

Buftea, Alberto Gabriel

May 4, 2023

Abstract

This document is created to help the design, plan and approach for a search relevancy product case. We will discover that in order to ensure relevant results for the users, first we need to understand what are their intent with the query, which is achieved with a process named query understanding. This process facilitates the retrieval of relevant documents, that have to be ordered form the most relevant to the least. We will explain how to first implement a tactical solution which does not need pre-ranked documents data for implementation so we can reach the constrained deadline, to afterwards explain the strategic solution consisting on a Learning to Rank Machine Learning approach that also considers user behavior as features. We will finally review how to ensure that the system delivers successful result to lastly explain what metrics would include a KPIs dashboard for tracking product milestones towards senior management.

Index

1 What is Search Relevance? 4

2 How will we ensure search relevance within our product? 6

2.1 Query Understanding 6
2.2 Tactical Solution: Boosting Ranking Approach 9
2.3 Strategic Solution: Learning to Rank Implementation 11

3 How are we going to implement the above into production? 13

3.1 Query Understanding 13
3.1.1 Query Correction / Suggestion 13
3.1.2 Stop-Words Removal / Stemming 14
3.1.3 Query Expansion 15
3.1.4 Query Classification 16
3.1.5 Query Tagging 17
3.2 Results Ranking 17
3.2.1 Tactical Solution: Boosting Ranking 17
3.2.2 Strategic Solution: LTR Ranking 18

4 How will we ensure appropriate results? 19

4.1 Model Drift 21
4.2 Senior Management Dashboard 22

1 What is Search Relevance?

Search relevance is the process of retrieving information that satisfies users needs based on the query that is launched. The query is a set of words written in a text box.

In [1], we read a user analogy who was trying to buy a manual lawn mower with no power source and was not able to find one online. He finally found one at gardening shop asking the sales person for help, who let him know the exact name of the tool he was looking for. Here, search relevancy is described as the practice of turning a search engine into a smart sales associate.

The trick to relevancy is that search engines like Solr and Elasticsearch are just sophisticated text matching systems. They can tell you when the search word matches a word in the document but they are not nearly as smart or adaptable as a human. Once a match is determined a search engine can use statistics about the relative frequency of that word to give a search result a relevancy score. According to the author, to develop a good search relevancy we should focus on those factors:

- Text Analysis / NLP
- Query Weights and Boost. Re-weighting the importance of various fields based on search requirements.
- Phrase/Position Matching. Requiring on the appearance of the entire query based on the position of words.
- Tags and Ontologies. Understanding the query and documents in terms of specific contexts instead of simply matching strings.
- Statistical Processing. Understanding the relationships between different words. For example, detect that frying eggs and spatula has some level of association.
- Search Engine Plugins to modify the built-in scoring to create a custom relevancy algorithm.
- Genetic algorithm, determine correct values for weights and boosts that produce the optimal result using a genetic/evolutionary process.

Achieving successful results is a complicated task because relevance is impacted by several factors which could vary from user to user. The factors on which results are ranked for an e-commerce website are not the same factors that should be considered when ranking for investment analysis. In addition, different users formulate different queries at searching the same context, likewise, the exact same query ideal relevance rank might be different for distinct users.

As [2] states, humans search for Things not for Strings. Andreas suggest finding the user intent within the query in order to encourage searchers to express their specific needs by driving them to create more specific queries. This is achieved in 3 stages:

1. Performing query expansion to improve user queries themselves. Handle typos, misspellings, term decomposition, stemming and lemmatization.
2. Understanding the user intent/meaning with the query. Extract mentioned entities and concepts form the query to predict one or more possible intentions with a certain probability.
3. Continuously test and learn what queries perform best for a given intent. Important to know that right intent may be dependent on time, region and other variables.

In [3] we read an interesting idea which states that users information needs are almost always incomplete, making the queries they enter imperfect. Users gain awareness of what they want as they search the information. Relevance is described as the process of helping users in the forwarding process of being more aware of what they really want, they start with an imperfect notion of what is needed and gradually getting closer to the exact question they want to ask.

In order to aid users in the process, he states the importance of the results having different options that contrast with each other, helping users take the next decisions in the forwarding process and move closer to what they want. For example, a passive job seeker will want to know that data science jobs are about python and statistics, while software engineer jobs are about .net and java. However, once they committed to a job, they still want to contrast details for example a job with no commute and another with a higher pay but which requires a long drive, both are relevant. Highlighting the differences is often more useful than similarities to a user making a decision. Users will not feel the results are as a whole if the query returns 5 identical items.

2 How will we ensure search relevance within our product?

Once reviewed the topic of search relevance and what are the main techniques used by other products to ensure they return results that are relevant for the users, let us now decide what should we do to ensure relevance within our product. The search relevance of our product will be divided by two different modules, a query understanding and a document ranking module.

In order to ensure relevant documents are being returned from the database based on the user query, we will first tend to understand what the user really wants by applying text processing, spell checking, query tagging, query classification and query expansion. After retrieving the relevant documents, considering the constrained deadline to get into production and the fact that we do not have user behavior because the product has not been released yet, we will follow a dual strategy for ordering the retrieved documents. We will first focus on a **tactical solution** which consist on boosting the default search engine rank, that is usually the BM25 algorithm, applying the boost over user profile parameters and smart annotations. While this system is deployed, we start collecting real time user behavior like clicks, time spent reading specific documents, downloads, past queries logs, and use that information to expand our product capabilities by implementing the *strategic solution* which consists on a Learning to Rank (LTR) listwise ranking approach.

Let's now review each one of the discussed topics in the following subsections.

2.1 Query Understanding

If the returned documents based on the user query are not appropriate, it does not matter how good is the ranking algorithm. User expectations will hardly be satisfied since we would not be retrieving the correct information. Therefore, in order to ensure that the search engine returns relevant documents based on the user input we have to focus on understanding the query and his intention. The following topics need to be considered:

- **Text processing the query.** This involves applying NLP techniques to ensure the statement made by the user is clear. We will apply three techniques to our user query as described in [5].
  - Spell Checking: Consists on suggest corrections for the words written by the user that are not in the spelling dictionary. Suggestions are found by comparing the misspelled words to the words

that we have in the dictionary using a similarity measure. Most commonly, the number of permutations of characters needed to correct the word is applied as measure. The suggestion is the word which requires less permutations to reach the suggested word starting form the misspelled word. In order to consider the query context at spelling suggestions, the noisy channel model is applied, where probabilities of words are calculated capturing the frequency of words in text and the probability of observing a word given that another word has just been observed. In this way, we will be able to suggest query corrections even if a word which has been correctly spelled does not belong to the phrase. For example, the query "think fish" does not have a spelling mistake, however, the probability of the word *think* given the word *fish* is much lower than the probability of the word *tank* given the word *fish*, $P(think|fish) < P(tank|fish)$, thus, we could suggest the user to introduce tank instead of think.

– Stop-words removal. Those are words that have little meaning apart form helping humans at describing nouns in text and representing relative position between two nouns. The words are mainly determiners like - *The*, *a*, *that*, *those* - and prepositions such like *over*, *under*, *above*, *below*. These words rarely indicate anything relevant about the document on their own. In information retrieval systems those words are called stop-words because the text processing stops whenever one is found and removes it from the list. Removing those words helps at system processing and increases retrieval efficiency. Constructing a stop-words list must be done with caution because adding too many stopwords to the list may result in bad information retrieval. This subject has been deeply studied and a variety of open source frameworks are available for performing this processing.

– Stemming. Part of the expressiveness of natural language comes from the huge number of ways to convey a single idea. This can be a problem for search engines, which rely on matching words to find relevant documents. Stemming reduce the different forms of a word like plurals and derivations to a common stem. Then, even though the query has the word "swimming" but the documents we have "swam" or "swim", by stemming they are all transformed into the same form, "swim", and so there is a match when searching for the term. At performing stemming we need to be careful since for some cases stemming will reduce the accuracy of the results. For example the query "fishing village" has a different meaning than "fish village", even though most of the stemming algorithms will reduce the word *fishing* to *fish*. This issue will be addressed through the query expansion.

- **Query tagging.** The process consists on detecting candidates named entities related to the user query and linking them to the pre-defined categories in the smart annotations such as companies, people, location, etc. Semantic tagging consist on identifying the semantic concept of a word or phrase and associate that with the semantic concept of the documents. As described in [6], detecting entities is a non trivial challenge since there are ambiguities of a word usage deepening on the context. For example, a query may refer to ”Washington” as a location while the same term may refer to a person in the query ”Denzel Washington”. Another common challenge is at setting boundaries, for example, the query ”New York” should be treated as a whole word and not two individual words ”New” and ”York”. The author states that we can find three different ways of tackling this challenge, design a rule based approach, a dictionary based approach or use machine learning. As for the first two, it may get too complex since the first consist on defining hand crafted pattern matching rules that needs to be personalized for each domain while the second approach involves creating a dictionary which should dynamically be updated to avoid outdating. The dictionary approach could be feasible at RavenPack since the company already tags all the documents with events, sentiments, organizations, companies as it is released in press, thus, maintaining the dictionary should not be difficult since it could be automatically updated each time there is a change in the corpus. However, the author defines that machine leaning being the most robust approach to follow for this task. Assuming that RavenPack already extracts entities from text using machine learning, it should be easy to implement the same strategy at extracting the entities from the user query, we may even be able to reuse the model. In addition, there are open source ML frameworks designed for query tagging.

- **Query classification.** Query classification is one of the main drivers of detecting the intent of the user with the query. As stated in [7], this is a very challenging task because queries are short and detecting the proper user intent requires more background. Recent state of the art publications achieves better results by using deep neural networks in detrimental of the Naive Bayes derivations models that are normally used at this. In [8] we can have a vision of how query intent is being performed at *LinkedIn*. They say have implemented the first Bidirectional Encoder Representations form Transformers *BERT* model in production for query classification. Briefly checking the internet we can see this model is available in some open source frameworks thus we will decide to copy LinkedIn approach in section 4.1.2

- **Query expansion.** Is a technique that consist on expanding the search query to match additional documents, achieving this trough the use of synonyms and semantically related queries. The key to effective expansion is to choose words that are appropriate for the context or topic, of the query. For example, ”aquarium” may be a good expansion term for “tank” in the query “tropical fish tanks”, but not appropriate for the query “armor for tanks”. Query expansion techniques are usually based on an analysis of word or term co-occurrence, analyzing how likely is for two words to occur in a given text window. A common measure is the *Mutual Information* which deducts if the two words are independent or not. For example, the words ”fish” and ”automobile” would have a mutual information score close two zero while if the two words tend to co-exist like ”fish” and ”boat” the score will be closer to 1. A score of 1 means that the words always coexist while a score of 0 means they never coexist in the defined text window.

2.2 Tactical Solution: Boosting Ranking Approach

After ensuring the search engine returns documents and news which are relevant for the user, now we need to decide the order in which the documents are presented. There are unlimited ways of ranking the results, it can be done randomly even though most probably we will not be that lucky to achieve relevant order using this strategy, thus, we need to design a strategy for this.

Most search engines already returns the document ranked by default. As we see in [9] the results are ranked by default considering the frequency that the terms in the query appear in the document. The most common way of ranking documents by search engines is the BM25 algorithm whose formula is presented in the figure 1:

$$score = tfNorm \cdot idf$$

$$idf = \log\left(1 + \frac{docCount - docFreq + 0.5}{docFreq + 0.5}\right)$$

$$tfNorm = \frac{(k+1) * tf}{k * \left(1.0 - b + b * \frac{fieldLength}{avgFieldLength}\right) + tf}$$

Figure 1: BM25 Ranking Formula

Whats this formula has two factors, **IDF** which counts how often a term occurs inside the whole corpus and penalizing documents in which the term occurs more often. So if in a corpus of two documents the query term term occurs 30 times, divided in 20 times in the first document and 10 times in the second document, the second document will have a higher idf score. The factor **tfNorm** does the opposite, meaning that if the term occurs more often in that document, the score of that document will be higher, but this factor takes into consideration the length of the field, penalizing the score of fields with higher length. Meaning that if the term appears in the the title of the document, that document is probably more relevant than other docs where the term appears in the body but not the title.

We are going to assume that the search engine rank by default the returned documents using this strategy and we will apply boosting over the results based on user parameters and smart annotations.

To rank the retrieved documents, we can either implement a machine learning approach that is trained based on past queries and correct list of ranked documents (knwon as ground truth), either we design a formula based strategy in which we decide what factors should be prioritized and a ranking score is calculated for each document based on it.

Considering that we have no previous user behaviour data and we do not have a considerable amount of user queries on which to train a ML model adding up to the constrained time-frame, we decide that the best approach is to initially implement the formula based ranking solution to deliver a minimum viable product since it has proven useful on other products and it is very easy and quick to implement because the formula is already implemented in the search engine, we just need to consider what parameters to take into consideration for boosting the result. Boosting means that given a document with its rank score calculated by the formula, if one of the terms in the document matches the selected boosting parameter its relevance score will be multiplied by the boosting factor. This factor could be positive, increasing the relevance of the document, or negative, decreasing the relevance.

To achieve satisfactory results,user preferences and their area of expertise should be taken into account when choosing the boosting parameters. Ideally we will first have conversations with some of the future users to understand what specific needs they have at information retrieval. Some of the boosting factors which we can use are:

- Novelty. Positively boosting documents that are novel, since in financial markets we would always prefer to have the latest information for a topic.

- Location. We could positively boost documents that relates to the same geographical area of the user, or if we know what markets does he/she invest, boost documents related with that markets.
- Entities. In semantic tagging entities can be any smart annotation, although here I would like to suggest gradually boosting documents based on companies or persons name depending on the user query. By gradually I mean giving a higher boost to the exact math, but giving a slightly lower boost to related entities with the entity extracted form the query.
- *N-grams proximity.* Phrases are important in information retrieval. For example, given the query ”black sea” documents containing this phrase are more likely to be relevant than documents containing the phrase ”the sea turned black”. In applications with large collections of documents, we identify phrases in the documents using a much simpler definition of a phrase which is any sequence of n-words, also known as **n-grams**. N-grams are generated by choosing a particular value for n and then moving that “window” forward one unit. The most commonly a word N gram occurs, the most likely is it to correspond to a meaningful phrase in the language, thus, if any document matches the query n-gram is rank should be boosted.

2.3 Strategic Solution: Learning to Rank Implementation

Once the minimum viable product is up and running, we will have the possibility to extract user behavior information from logs and cookies. At this moment we can track their activity and use that information to train a machine learning model to automatically rank the results replacing the boosting strategy.

This decision was taken because machine learning requires a big amount of relevant data to train and test the model, data which will not have until users start interacting with our product. There is another approach that can be suggested which consists on supplying users with queries and different results ranking options based on the query so they can choose which one is more relevant for them. But this strategy has several problems, the main being that users do not judge equally when they are being given a ”fake” example compared to when they are in need of the information. In addition, this approach would require availability of users to answer our experiment. Considering the amount of information needed for training a machine learning approach it requires a lot of time to get that information, risking our constrained deadline and just to find that the examples are not accurate enough.

Learning to rank *LTR* is the application of machine learning, typically supervised, semi-supervised or reinforcement learning, in the construction of ranking models for information retrieval systems. LTR solves a ranking problem on a list of items, coming up with the optimal ordering of those items. As stated in [10], there are three main approaches when it comes to solving LTR problem. The point-wise approach whose input space contains the information of a single document and the output is the relevance of that document, the pair-wise approach whose input space contains the information of a pair of documents while the output represents the preference between the two documents scoring from -1 to 1, and the list-wise approach whose input space contains the whole list of documents related to the query and the output contains the either the relevance degree of each document or the ranked list for the documents.

Recent investigations proved that the list-wise approach achieves better results although is quite a recent technique and it is more complex at training. In [11] is stated that list-wise ranking are more suited for world wide web searches than enterprise search so that may make us doubt going for the list-wise approach but the article was published 5 years, computer power has increased and I am considering the two factors below to decide experimenting further with the pointwise approach for the strategic solution:

- Once the application will be in production we will be able to collect user information such as the documents they visit, how long do they spent reading the documents, which position in the rank had the first document they clicked on, in addition with other previously calculated features and smart annotation which we can use as model inputs together the user query. I consider that we will be able to supply the model with a vast amount of features such that it can extract strong relationships between inputs to predict the adequate rank at the output.

- Since we will already have the tie-breaking model in production, we gain time to experiment with the list-wise approach and if it does not stands out as appropriate we can replace with the pairwise without risking the integrity of our product and reusing the same features as inputs.

Based on previous previous investigations, LTR models that achieved better results are based on *ListNET* or *ListMSE* models. We can also find a recent article published in 2022 by [12] where they took a BERT model and trained it for list-wise search engine results ranking. They stated achieving good results wit a model which they called LisBERT deriving form the list-wise BERT. We will discuss more about this topic in section 4.4.2

3 How are we going to implement the above into production?

In this section we will discuss what technologies are open source available for implementing the techniques discussed under section 3. It will mainly consist on short explanations and code examples form official sources since it is not the scope of this project to create the code for achieving the solution.

3.1 Query Understanding

Figure 2 represents the query understanding module in which we can see every query transformation before retrieving the relevant documents form the database. Let us see an example of each one of the transformations in the subsections below.

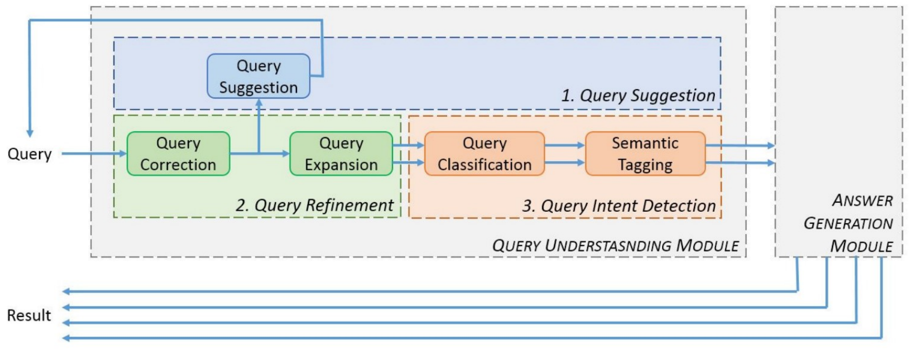


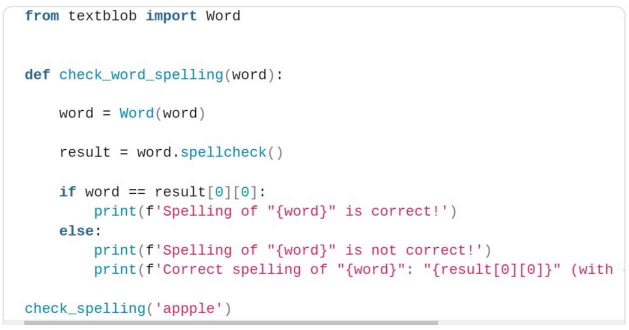
Figure 2: Query Understating Module

3.1.1 Query Correction / Suggestion

Query correction and suggestion comes together since once we detect there is a misspelling in the text introduced by the user we can suggest him to correct the word he has misspelled.

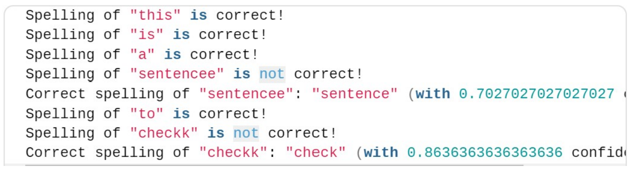
TextBlob is a python library which we can use at checking the user spelling mistakes. It returns the correct spelling for the word together with a calculated confidence level as seen in the figures 3 and 4.

In order to achieve the level of performance discussed in section 3.2 this technique need to be combined with the BagOfWords to extract proximity embeddings form words so we can detect incoherent queries even though



```
from textblob import Word


def check_word_spelling(word):

    word = Word(word)

    result = word.spellcheck()

    if word == result[0][0]:
        print(f'Spelling of "{word}" is correct!')
    else:
        print(f'Spelling of "{word}" is not correct!')
        print(f'Correct spelling of "{word}": "{result[0][0]}" (with

check_spelling('appple')
```

Figure 3: TexBlob statement



```
Spelling of "this" is correct!
Spelling of "is" is correct!
Spelling of "a" is correct!
Spelling of "sentencee" is not correct!
Correct spelling of "sentencee": "sentence" (with 0.7027027027027027
Spelling of "to" is correct!
Spelling of "checkk" is not correct!
Correct spelling of "checkk": "check" (with 0.8636363636363636 confid
```

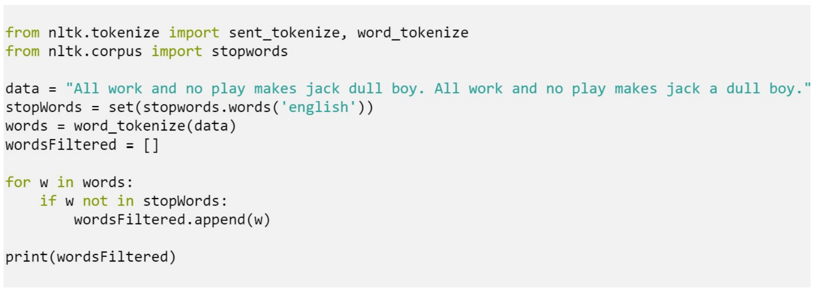
Figure 4: TexBlob results

there are not misspellings.

3.1.2 Stop-Words Removal / Stemming

We can use the *NLTK* ”natural language processing toolkit” libray in python for stemming and stop-words removal.

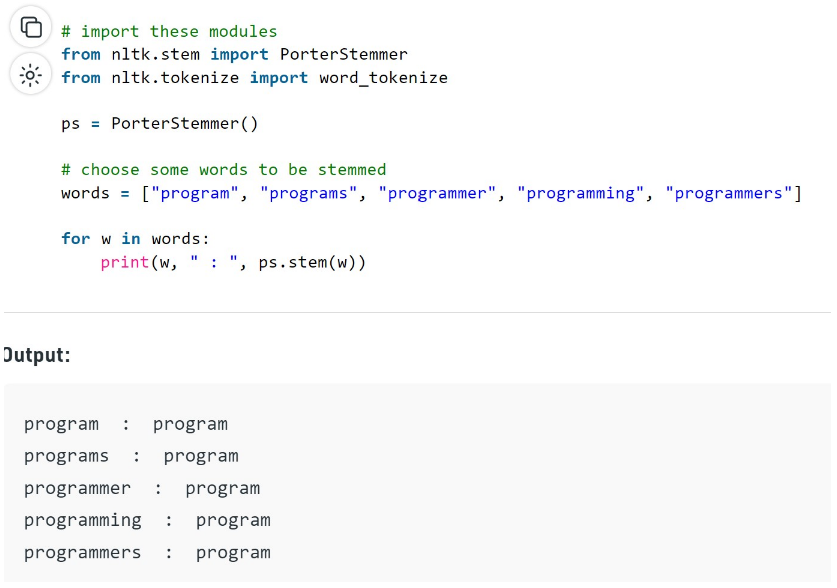
For stop-words removal the library includes a corpus of stop-words that can be imported from its corpus as in figure 5.



```
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords

data = "All work and no play makes jack dull boy. All work and no play makes jack a dull boy."
stopWords = set(stopwords.words('english'))
words = word_tokenize(data)
wordsFiltered = []

for w in words:
    if w not in stopWords:
        wordsFiltered.append(w)

print(wordsFiltered)
```

Figure 5: StopWords Removal

Similarly, for stemming we can use the same library as seen in the figure 6. As discussed in section 3.1, we could implement a controlling strategy to decide if it is applicable to apply stemming or not depending on the query context.



```
# import these modules
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()

# choose some words to be stemmed
words = ["program", "programs", "programmer", "programming", "programmers"]

for w in words:
    print(w, " : ", ps.stem(w))
```

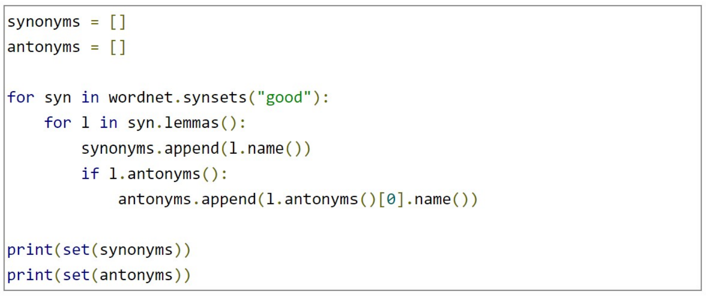
Output:

```
program  :  program
programs  :  program
programmer  :  program
programming  :  program
programmers  :  program
```

Figure 6: Stemming

3.1.3 Query Expansion

For query expansion we can use the *WordNet* python library which was created to find meaning of words, synonyms, antonyms such as can be seen in figure 7.



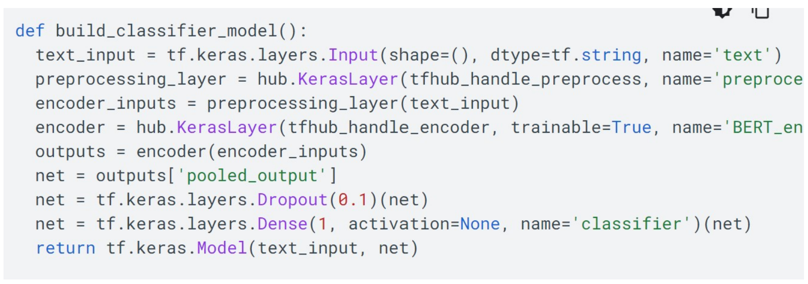
```
synonyms = []
antonyms = []

for syn in wordnet.synsets("good"):
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print(set(synonyms))
print(set(antonyms))
```

Figure 7: Synonyms / Antonyms finding

After finding the synonyms for the words in the query we have to apply synonyms into the documents retrieval process.

3.1.4 Query Classification

The BERT model is available with the python *TensorFlow* library which as themselves state, "BERT models are usually pre-trained on a large corpus of text, then fine-tuned for specific tasks". The library permits the definition of your own model as in the figure 8 which we will have to train for our specific domain.



```
def build_classifier_model():
  text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
  preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preproce
  encoder_inputs = preprocessing_layer(text_input)
  encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_en
  outputs = encoder(encoder_inputs)
  net = outputs['pooled_output']
  net = tf.keras.layers.Dropout(0.1)(net)
  net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)
  return tf.keras.Model(text_input, net)
```

Figure 8: BERT Model Creation

The BERT pre-trained layer will be a hidden layer of a classification model which have the structure shown in figure 9. The query text should be pre-processed before feeding into the BERT model, although tensorflow already provides a pre-processing layer for each one of the available BERT models, it would be interesting to manually pre-process the text using Bag-Of-Words technique which will create embeddings that are similar to each other for similar words which will be beneficial for the model training.

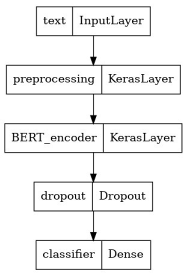
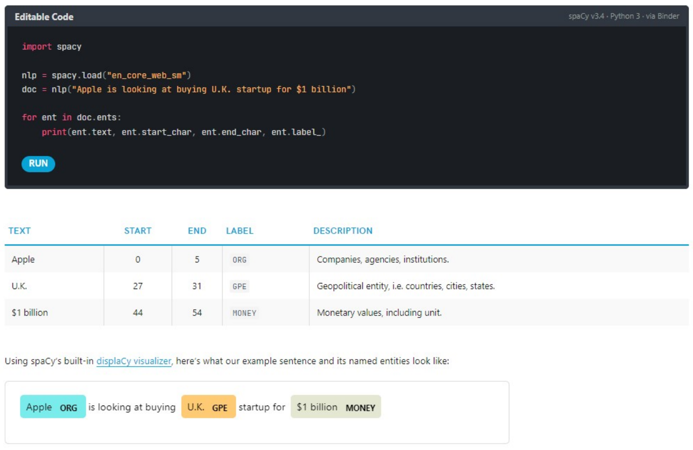


Figure 9: BERT Model Structure

When it comes to data collection, we first need to analyze the different types of information which our users consumes and decide how many categories we want the model to predict. It can be as simple as having the model deciding between one of the investing sectors such that we already know what documents are related to that sector form our smart annotations. In order to form the training data, examples of user queries related to each sector need to be created. Even though at this stage the application is not being used thus we do not have a log of user queries, those information is easier to build based on experts, open source available data and even other user data which we could store in our systems form which we can extract their needs, formulate queries and assign it with one investing sector such that we use it for training the model. This statement lays here to point out that artificially forming the training data for this classification model is much easier and more accurate than artificially deducting the order of the documents which would be relevant, thus, we could afford implementing this model into production with the tactical solution.

3.1.5 Query Tagging

spaCy is an open source python library that enables advanced natural language processing functions. It has an entity recognition built in module which given a text extract the entities as seen in figure 10.



```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

TEXT	START	END	LABEL	DESCRIPTION
Apple	0	5	ORG	Companies, agencies, institutions.
U.K.	27	31	GPE	Geopolitical entity, i.e. countries, cities, states.
$1 billion	44	54	MONEY	Monetary values, including unit.

Using spaCy's built-in displaCy visualizer, here's what our example sentence and its named entities look like:

Apple ORG is looking at buying U.K. GPE startup for $1 billion MONEY

Figure 10: Entity Recognition

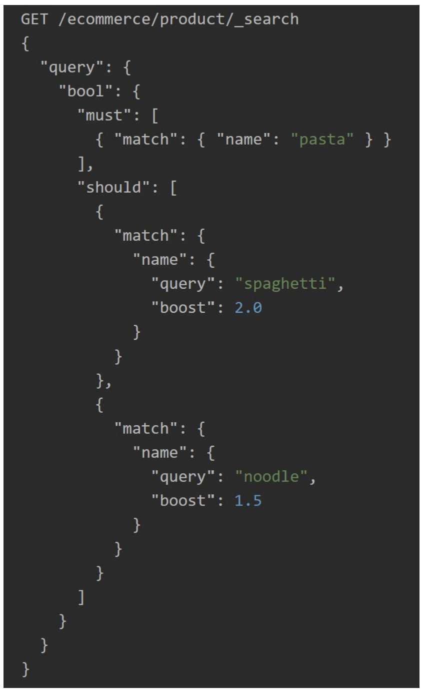
3.2 Results Ranking

Now we are going to briefly review how a ranking algorithm such as the tactical solution and strategic solution can be implemented. As stated above, I am not going to code any solution just briefly stating the available technologies.

3.2.1 Tactical Solution: Boosting Ranking

When it comes to boosting is easy and quick to apply, for example the *ElaticSearch* engine allows boosting of documents in top of the pre- defined

BM25 algorithm if the document matches the words that you decide to boost. This can easily be applied as shown in figure 11.



```
GET /ecommerce/product/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "name": "pasta" } }
      ],
      "should": [
        {
          "match": {
            "name": {
              "query": "spaghetti",
              "boost": 2.0
            }
          }
        },
        {
          "match": {
            "name": {
              "query": "noodle",
              "boost": 1.5
            }
          }
        }
      ]
    }
  }
}
```

Figure 11: ElasticSearch boosting

Here we may need to create additional software to automatically decide what terms to boost, including a higher boost for exact matches while a lower boost for synonyms or related entities. The above explained *spaCy* library will come to be helpful at creating this module since it permits a range of NLP features such as N-grams and part of speech (POS) tagging.

3.2.2 Strategic Solution: LTR Ranking

As discussed in section 3.3, the listwise approach is the state of the art document ranking machine learning approach. As can be seen in [9], to create a LTR model the following steps needs to be tackled:

1. **Create judgment list (Ground Truth).** This consists on getting ranked documents based on user queries which we know are perfectly ranked. In our case we will analyze user behavior to decide what is the most accurate ranking for the list of documents. We track on what documents the user clicks when presented with the boosting approach rank, we track how long do they spend reading the document. using

this information we can built the ideal ordered list based on the user query, such we built the grand truth.

2. **Define features of the model.** Features are the model inputs togheder with the model. In this case, we can use the list of items which we applied boosting to as features. So smart annotations such as document novelty is going to be an input feature for the LTR approach. We can also use dynamic features such as how many times a document has been clicked.

3. **Log features during usage.** This steps consist on arranging the features for each user query that we are going to train in such a way that it can be inputted into the model.

4. **Train and test the model.** This step consists on normalizing the features, splitting between train, test and validation sets and starting the training process.

5. **Deploy the model.** Consists on deviating the raking system in production form the boosting approach to the LTR approach. So at this step we decouple the tactical solution and start with the usage of the strategic solution.

6. **Feedback loop.** This consist on keeping track of the user behavior while the model is in production to deduct if the model prediction is as good as expected.

When it comes to implementing the list-wise approach document ranking, the *TensorFlow* library offers the recommenders framework which is designed for building recommender system models. We can see an example on how to implement a listwise raking system in the following url: $https://www.tensorflow.org/recommenders/examples/listwise_ranking$.

4 How will we ensure appropriate results?

The system is being designed for the users, thus, the most accurate way of deciding if the ranking system is successful is by checking what documents the users clicks based on the results that are being presented. If most of the users switch to the second page of the search results without clicking or just briefly checking results on the first page to quickly return back, we can ensure the system is not successful. On the opposite side, if most of the users deeply reads almost all of the results on the first page, even downloading some documents, and barely changing to other pages just after staying for a decent time studying the results of the first page then we can ensure our system is successful.

The problem with the ideal description of success above is that in reality we will never have such extreme negative or extremely positive situations, so tracking users behavior is useful specially to prevent reaching the extreme negative and getting the system as close to the extreme positive as possible, however, users can only see the information which is being presented in front of them thus there's also a bias in the system since probably no user will move to the 5th page of the results, they better give up searching rather than checking 5 pages of results, but, how do we know that results at 5th page are not relevant?. Furthermore, if two results are presented on the first page ranked differently and most of the users click on both of them, how do we really know that the higher ranked result is certainly more relevant to the user than the lowered ranked result?

There are ways of checking the above such as randomly (for the user side) inserting a non relevant document (one which would be shown in the 5th page for example) in the first page results, just to ensure that no user clicks on that document, they skip it. This is one way of ensuring that the model is performing well at ranking. Another approach, consists on randomly taking a pair of documents form the first page of results and flip their position, for example we flip the 2nd ranked document with the 5th ranked document. If in this case the majority of the users still click before on the document whose position should have been the second (they click before on the 6th document rather than on the 2nd), is a measure that our ranking strategy performs well.

This measure of how well the results are ranked based on user expectation is a measure of the effectiveness of the model and some metrics have been introduced in order to quantify them. The most typically used are *recall*, *precision* and *fallout*:

- Recall is the true positive success rate representing total number of relevant documents retrieved by the search from the total existing relevant documents.
- Precision is positive predicted value representing the number of relevant documents retrieved form total number of retrieved documents.
- Fallout represents the proportion of not relevant documents retrieved form the total number of documents.

The above are metrics which can be used to measure the overall effectiveness of the system, we should also measure the efficiency by tracking the processing time and query latency. Although this is an specialized product for financial analyst so we may prefer effectiveness over efficiency, for a financial analyst searching for confidential information that he cannot find anywhere else, is more important to get the information he need accurately than getting it in a matter of second, we cannot allow the system to take to the order of tens of second to retrieve the information since it the user experience would be extremely negative. So we mus t ensure returing the results of the query in a matter of several seconds as maximum.

4.1 Model Drift

In addition to measuring the success of our system based on the data which we make available for the user, we need to pay special attention to environment changes since the appearance of new categories not modeled on our system could be ignored resulting in relevant documents not being showed to users, what's more, the searching system may even don't realize the existence of those documents. This is known as model drift, and is defined as the degradation of a model prediction power due to changes in the environment affecting the relationships between the variables.

An example of this would be the change of the CEO of a company, if the model has learned to link "Microsoft" with "Sataya Nadella" but suddenly the CEO of Microsoft changes to "Alberto Buftea", will the model correctly rank a relevant document with Alberto's name related to a Microsoft search?

Those systems vulnerabilities must be taken care of by implementing an external tracking system which detects changes in the environment. Usually, for a classification model we would realize such type of changes because the output probability of the model wouldn't be confident enough for any of the classes, in contrast, our system ranks a list of documents form most relevant to least relevant assuming whatever is retrieved by the search engine so an internal threshold for detecting drift would be very difficult to implement.

At RavenPack, since we already tag new incoming documents with smart annotations it should not be difficult to detect environmental changes and test the model with user past queries form the log to compare the results.

Another indicator which I am thinking of is an unusual reduction of users using the system. If we detect that for a longer period of time our product has less demand and it is not on specific dates such as Christmass or Summer, then we may assume something to worry about has happened.

Lastly, I would like to comment in this section an interesting approach called *interleaving* seen at [13] which consists on intersecting two models ranking results as seen in figure 12. This approach would let us know what model do users prefer based on the documents they click form the results page.

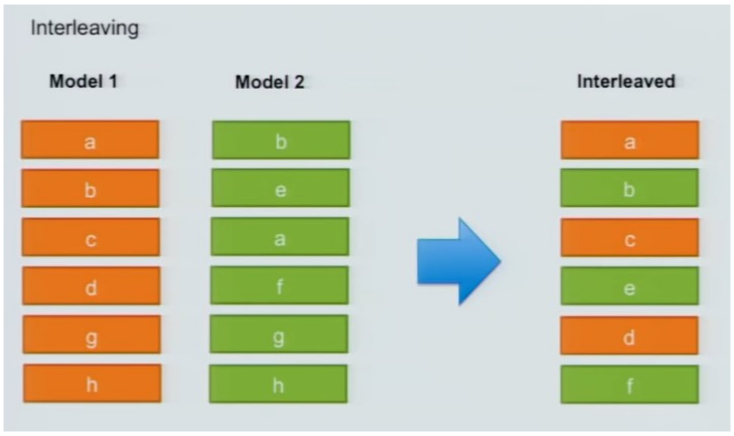


Figure 12: Interleaving - Intersection of two models ranking results

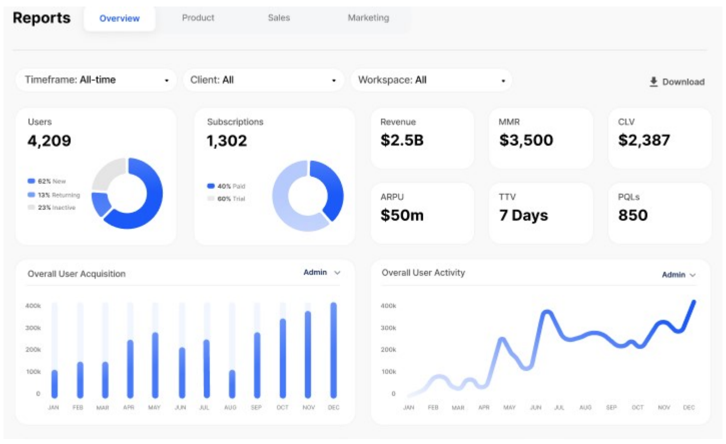
I see that we can benefit in three different ways form the interleaving approach because it gives us a metric to measure how well the principal model is preforming compared to other model thus we have a model metric, it advises us of possible model drift if users seems to prefer the output of the secondary model which has been trained with recent data and it allows us to test the effectiveness and efficiency of model upgrades without risking its direct transition to production but a gradual transition instead, lowering the risk of having an incident.

4.2 Senior Management Dashboard

The whole point of designing such a product is delivering unique features to our users standing out from our competence. If the company decides to invest in this product is because we need to see returns. The best way to present metrics to senior management is to measure the returns and present them in a clear visual way, easy to understand and easy to deduct what's happening so decisions could be taken form them. Special attention has to be paid on the fact that senior management goal is to take business decisions, therefore, whatever dashboard / metrics we design form them, they should meet the function of assisting this decision taking.

In general, most relevant metrics for senior management are called KPIs "Key Performance Indicator" which are metrics used at evaluating if the organization is meeting its objectives. The dashboard should be easy and intuitive such as seen in figure 13, below I am stating some metrics which should be introduced and presented graphically.

- Percentage of goals achieved based on initial plan.
- Rate of users increases since last milestone.
- State next steps, expected deadlines and expected results.
- Identify future threats / stoppers.
- Amount of budget spent versus expected expenditure.



Timeframe: All-time
Client: All
Workspace: All

Users 4,209
62% New
13% Returning
23% Inactive

Subscriptions 1,302
40% Paid
60% Trial

Revenue $2.5B
MMR $3,500
CLV $2,387
ARPU $50m
TTV 7 Days
PQLs 850

Figure 13: Example of KPIs dashboard

References

[1] What is search relevance? - opensource connections - https://opensourceconnections.com/blog/2014/06/10/what-is-search-relevancy/.

[2] (60) humans—search for things not for strings | linkedin - https://www.linkedin.com/pulse/humans-search-things-strings-andreas-wagner/.

[3] What is a ’relevant’ search result? - opensource connections - https://opensourceconnections.com/blog/2019/12/11/what-is-a-relevant-search-result/.

[4] How algolia tackled the relevance problem of search engines - algolia blog | algolia blog - https://www.algolia.com/blog/engineering/how-algolia-tackled-the-relevance-problem-of-search-engines/.

[5] Trevor Strohman W. Bruce Croft, Donald Metzler. Search engines - information retrieval in practice. *Pearson Education*, 2015.

[6] Paul Ashaolu. Applying machine learning algorithmsfor named entity recognition. *Universitat Polit‘ecnica de Catalunya*, 2014.

[7] Pushpak Bhattacharyya Pankaj Singh. Survey of query intent detection. *Indian Institute of Technology Bombay*, 2020.

[8] Huiji Gao Bo Long Xiaowei Liu, Weiwei Guo. Deep search query intent understanding. *LinkedIn*, 2020.

[9] Learning to rank search results.

[10] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends inInformation Retrieval*, 2009.

[11] Kristofer Tapper. Learning to rank, a supervised approach for ranking of documents. *Chalmers University of Technology*, 2015.

[12] Sagnik Sarkar Lakshya Kumar. Listbert learning to rank e-commerce products withlistwise bert. *Myntra Designs Pvt. Ltd*, 2022.

[13] Search quality at linkedin.