Industry Classification @BlueVine

OPEN DATA SCIENCE CONFERENCE

SAN FRANCISCO: NOV 2-4

Presentation Overview

- Who we are
- Why do we need industry classification?
- What we have today
- Development process of new model based on web-data
- Preliminary model performance metrics
- Main challenges & example of interesting use-case
- Key Takeaways

Presentation Overview

LINKS TO PRESENTATION AND CODE

- BlueVine Data Science Blog (coming soon!):
  https://medium.com/bluevine-data-science

- BlueVine public Git repo (coming soon!):
  https://github.com/bluevine

BlueVine

WHAT DO WE DO?

- Mission: Offer convenient and flexible access to working capital for small and medium sized businesses.

- Current products:
  - Revolving Line of Credit
  - Invoice Factoring (receivables backed financing)

- On track to fund over $500M by end of 2017.

BlueVine

- ✓ **24 hr. approvals**
- ✓ **100% online**
- ✓ **On demand**
- ✓ **Simple pricing**

Industry Classification

WHY IS IT IMPORTANT?

- We want to avoid certain industries altogether (e.g. adult entertainment, weapons, gambling).
- Understand our portfolio (e.g. react to crises in certain industries).
- Understand our clients better (e.g. a client with a mid-range credit score could be excellent compared to his industry).
- Identify potential fraudulent behavior (e.g. a pet shop working with a law firm).

Industry Classification

DEFINITIONS

List Size

- Big enough to cover important industries.
- Small enough to be feasible for a model.
- Final list has 50 industries.

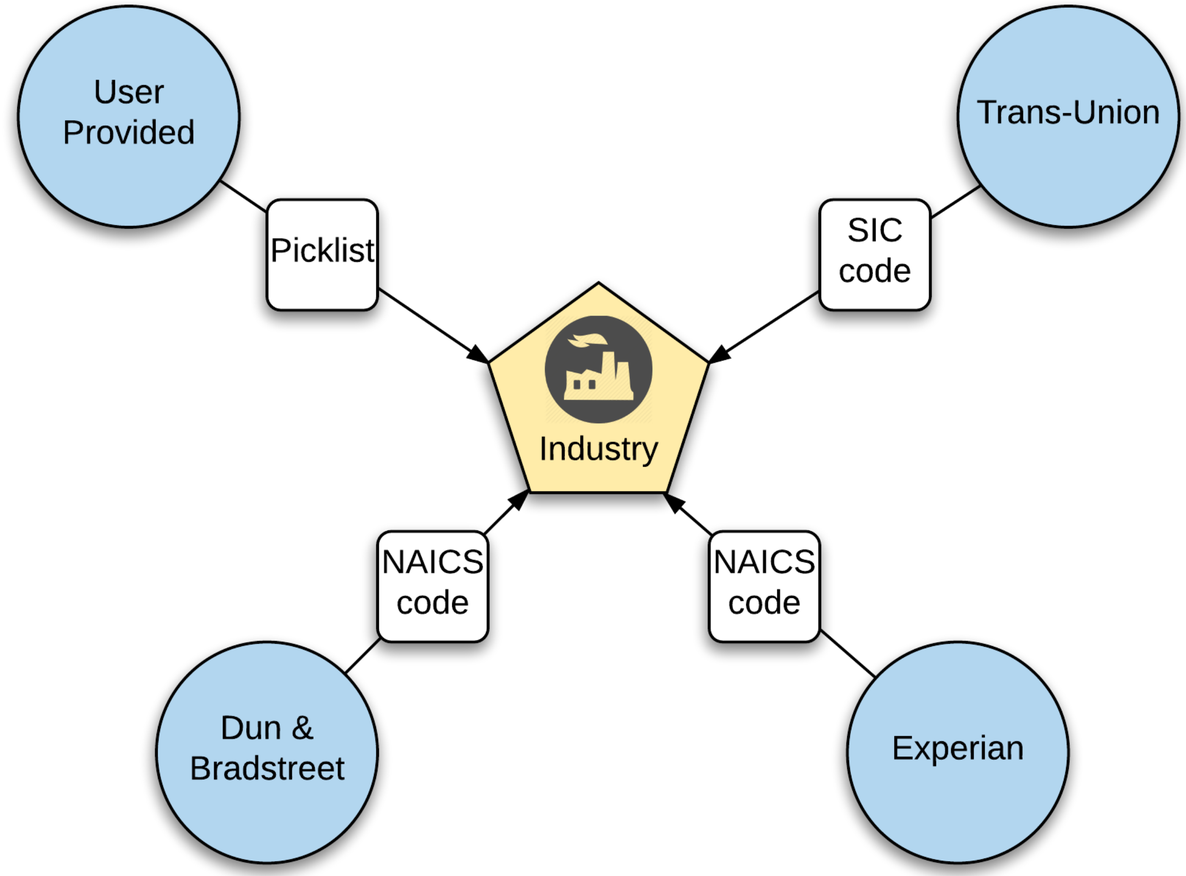
List Resolution

- High enough to make business distinctions (e.g 5 different types of construction)
- Low enough to be separable.

Data Sources

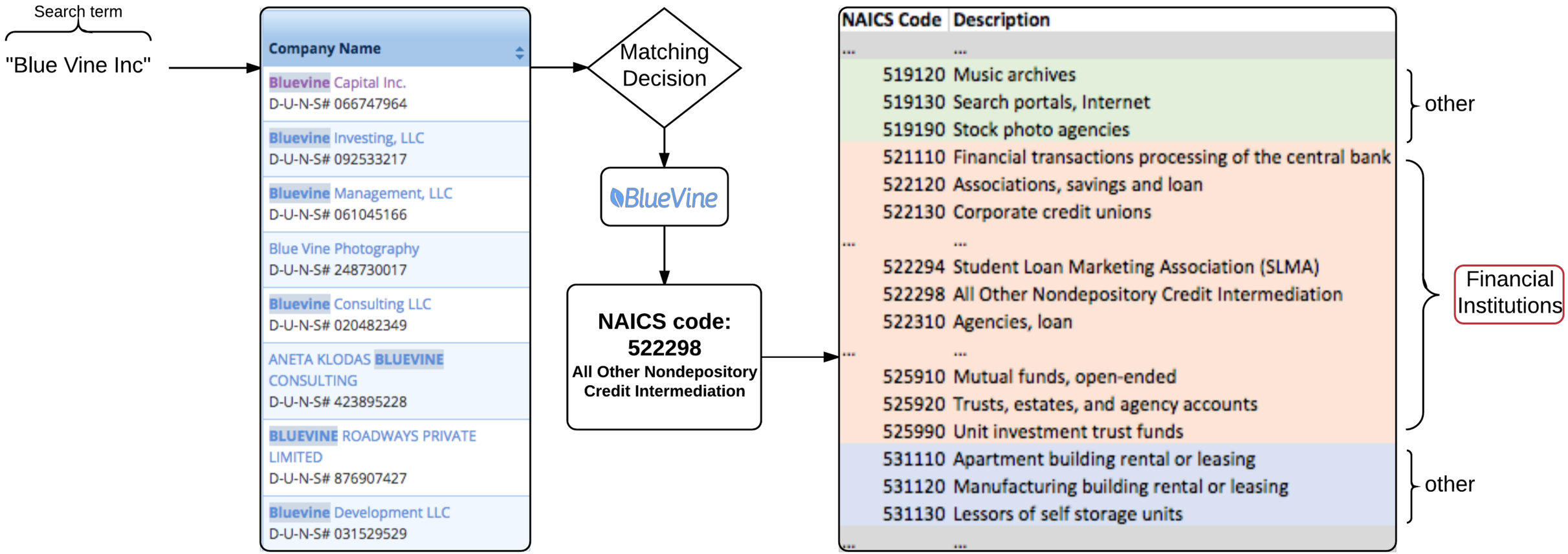
EXISTING SOURCES

- Industry data comes from multiple data sources.
- Each source has to be translated to be usable.



Data Sources

EXISTING SOURCES - E.G DUN & BRADSTREET

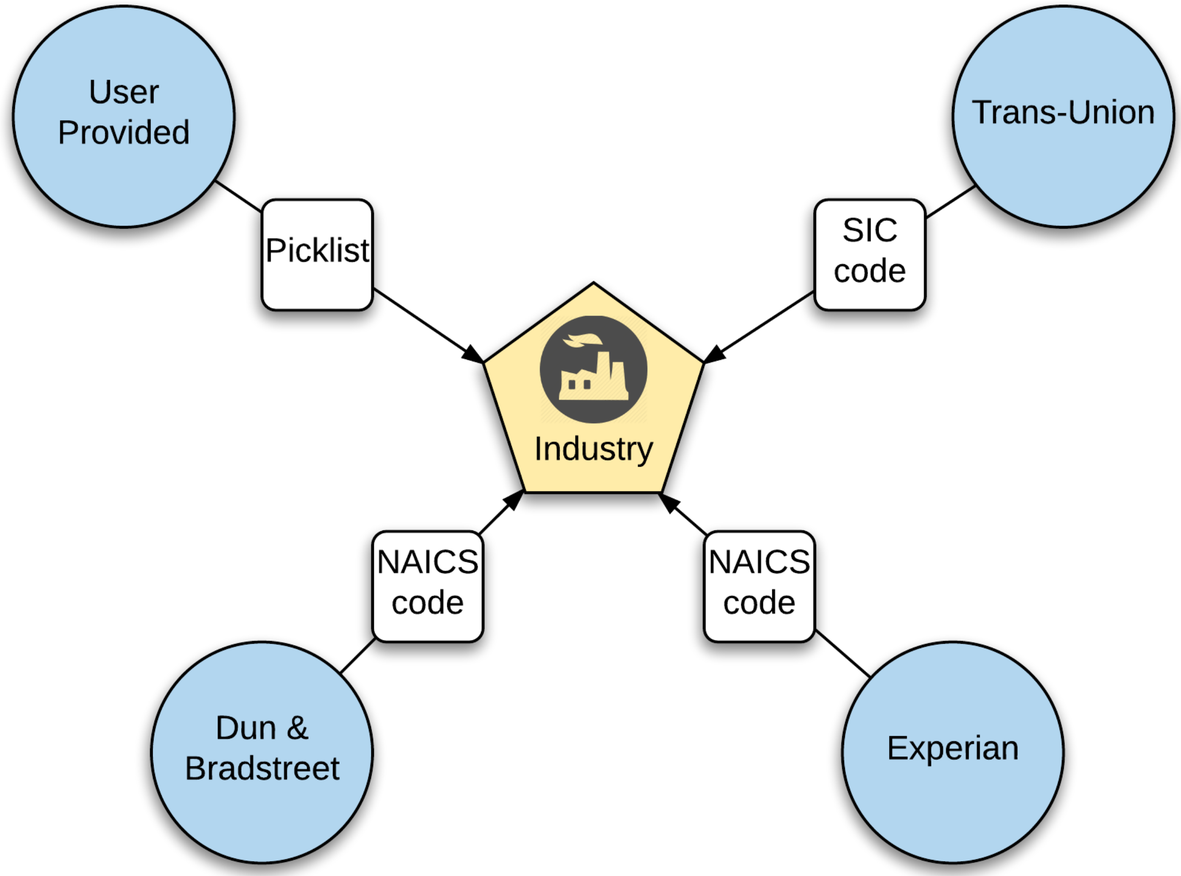


Data Sources

EXISTING SOURCES

Biggest problems

- Sometimes the data sources just don't return any data.
- Sometimes we fail to successfully translate the return data
- Overall there is a big coverage gap: 50%.
- Pricy $$$



Data Sources

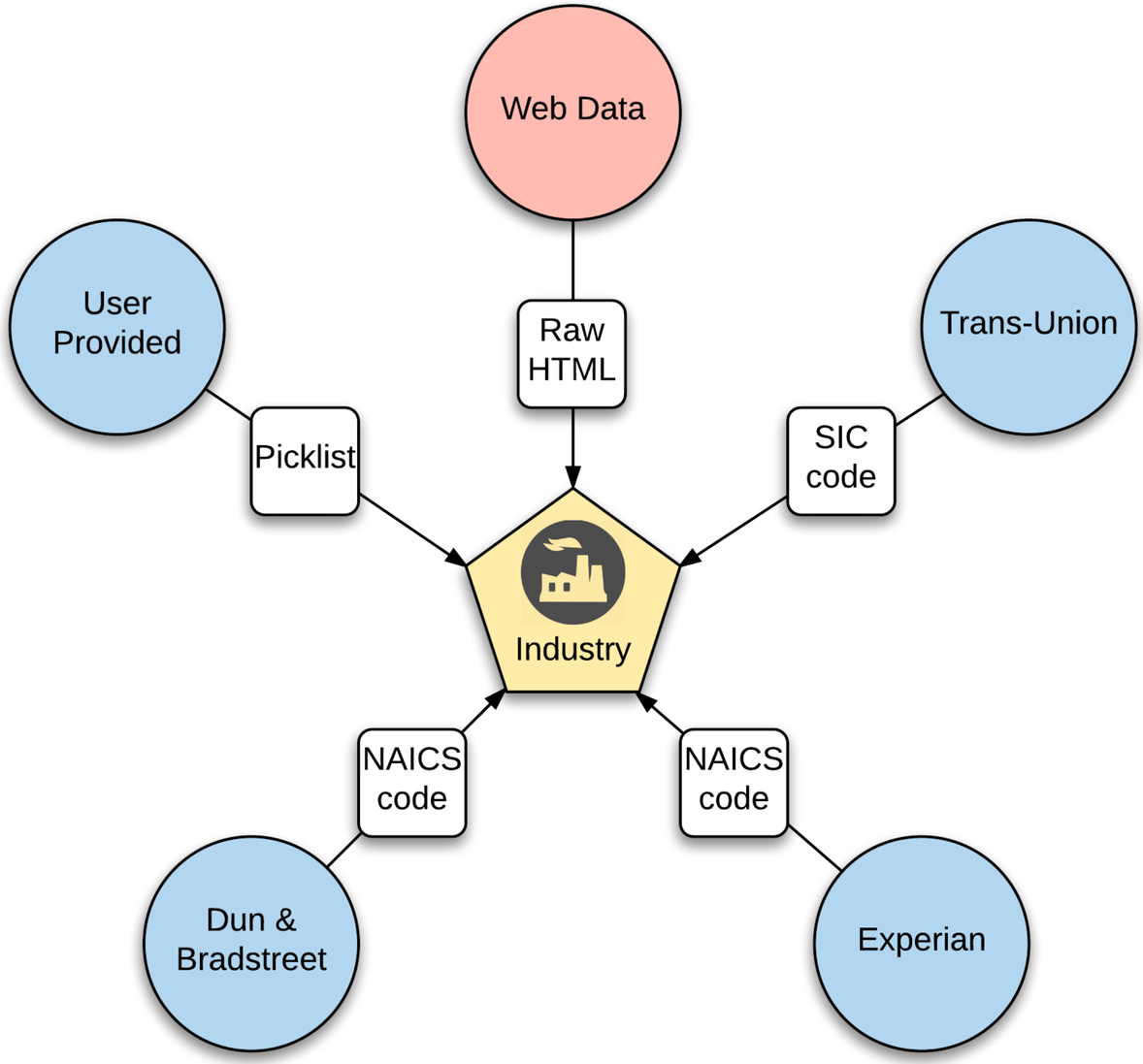
NEW SOURCE - WEB DATA

Pros

- Widely available: can close the coverage gap.
- Cheap: scalable price-wise.
- Easily accessible: scalable resources-wise.

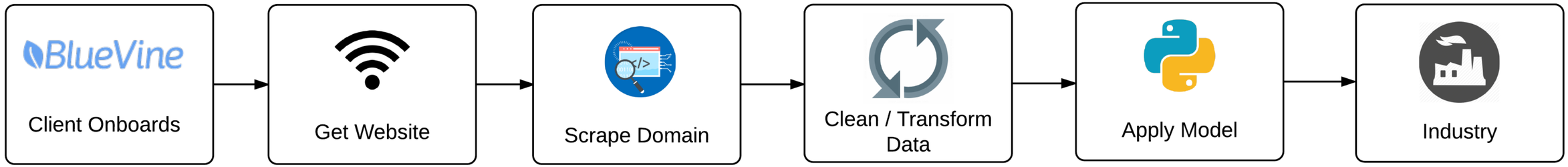
Cons

- Data is unstructured.
- Supervised implementation requires tagged set.



Data Sources

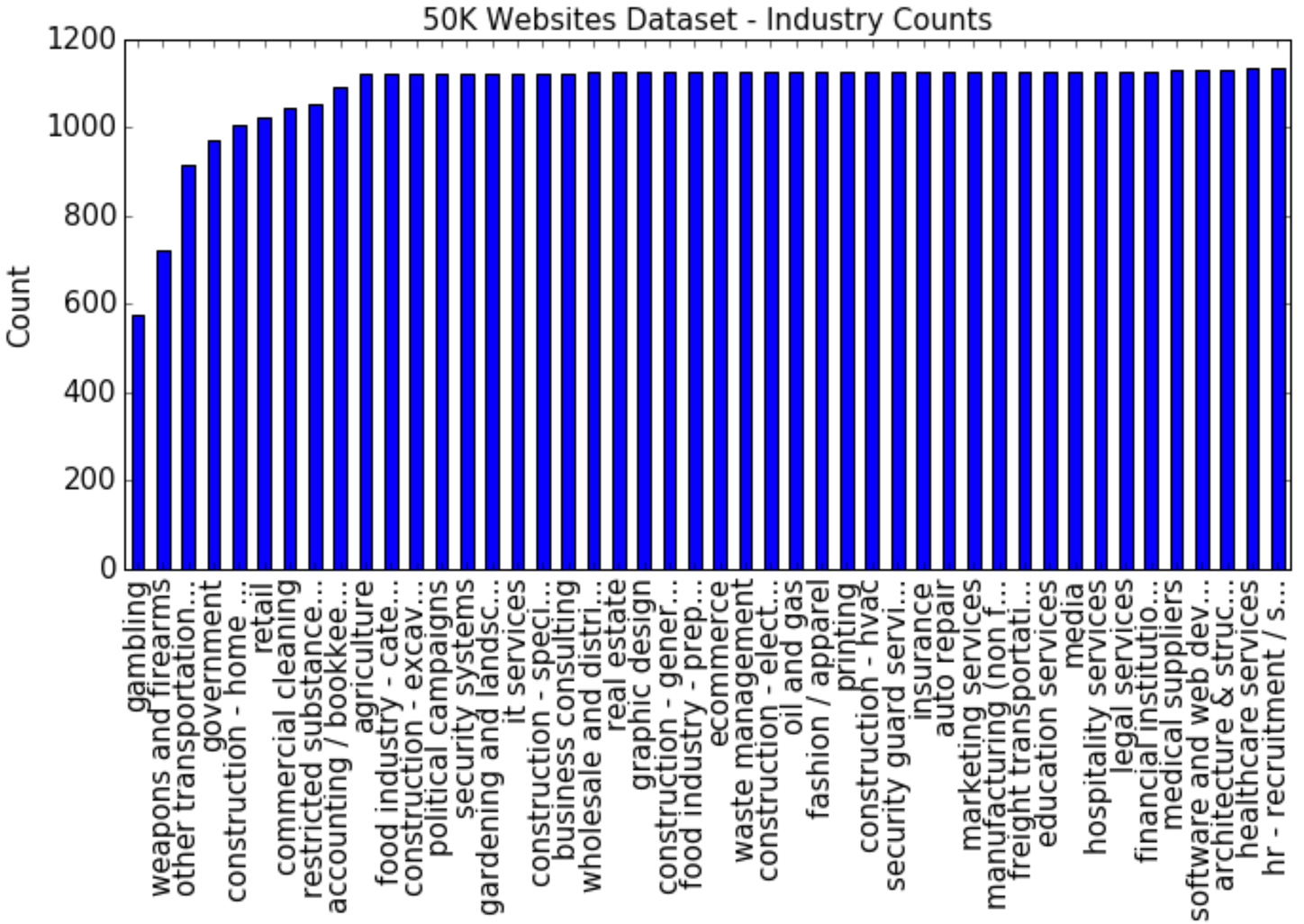
NEW SOURCE - WEB DATA - BASIC FLOW



Web Model - Tagging

THE DATASET

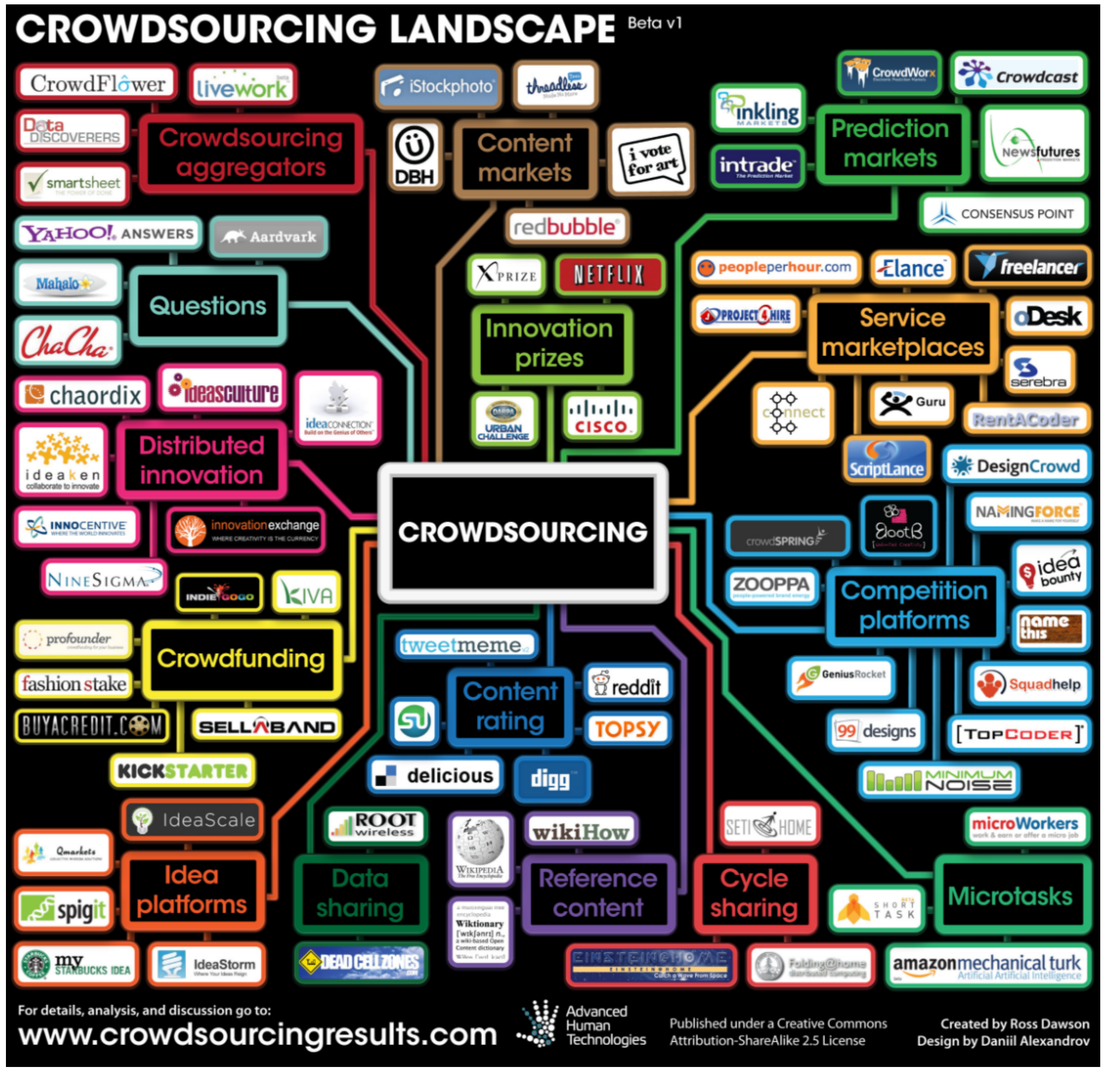
- 50,000 websites, ~1000 per industry.
- “Balanced” using D&B as proxy.
- Some supplemented using web searches (lists).
- Assumption: some will be easier than others.



Web Model - Tagging

IDEA

- Use a crowd-sourcing platform to perform the task of tagging the entire 50K row dataset.
- There are **MANY** to choose from →
- Our task is not a “micro-task”, so we require:
  - Platform flexible enough for complex task
  - Quality control mechanisms
  - Good price



Web Model - Tagging

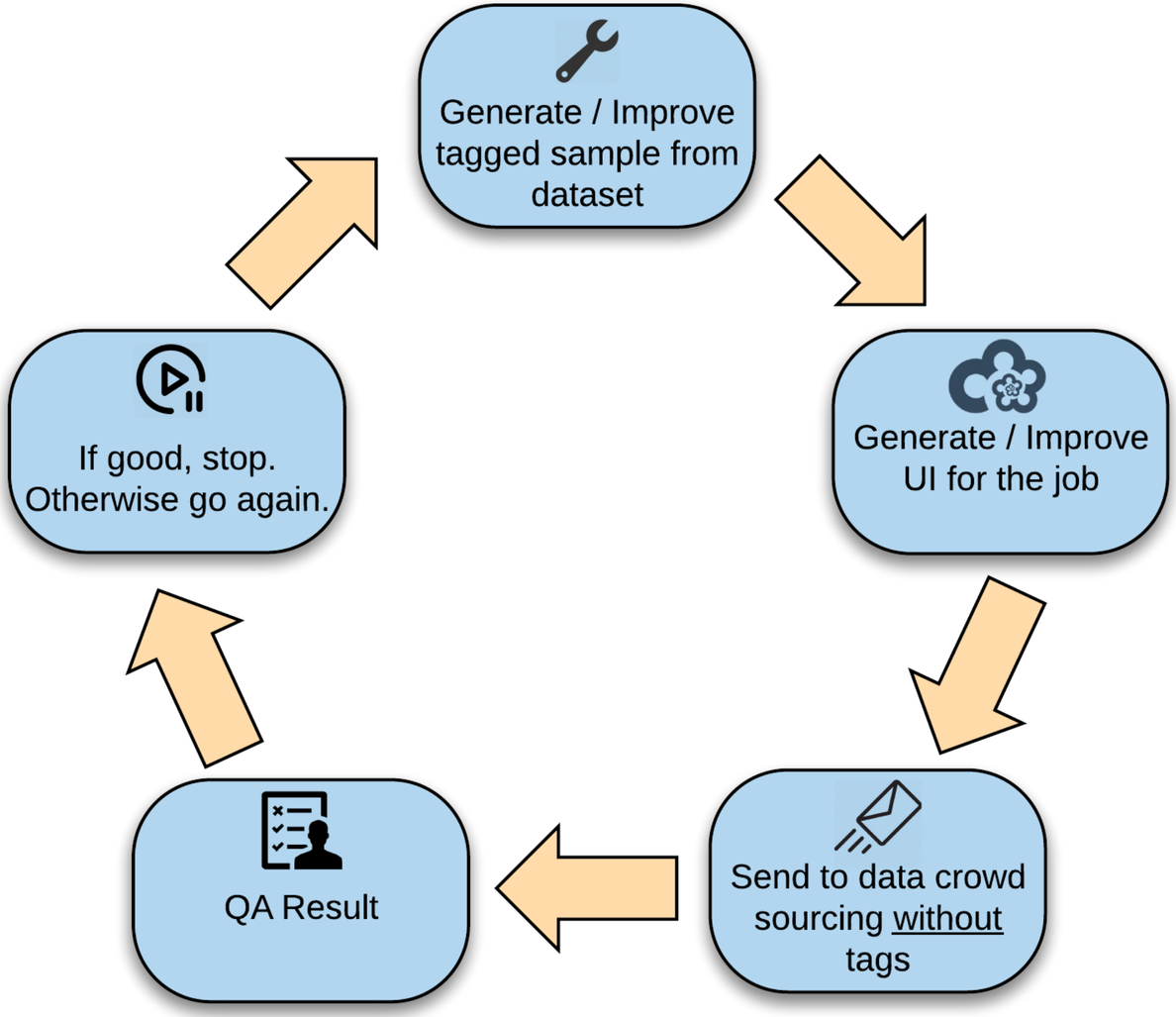
IDEA

- After some trials we decided to go with CrowdFlower.
- Selling point #1: A specific interface is created & tailored to our specific job.
- Selling point #2: Effective quality monitoring and control mechanisms.
- Selling point #3: Good customer support.

Web Model - Tagging

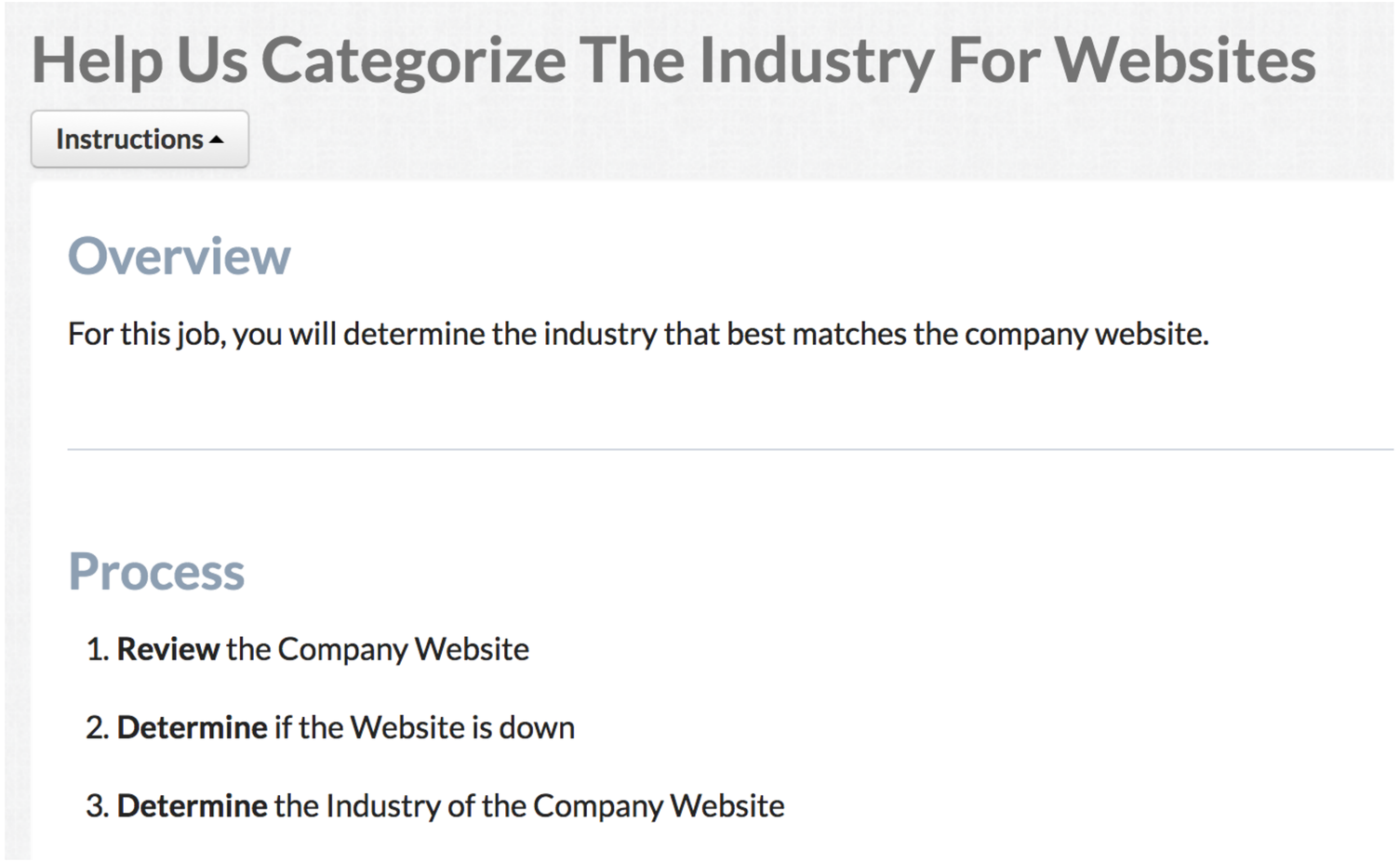
PROCESS

- Manually tag a small subsample of the dataset — 2000 rows, approx 40 websites per industry.
- Send that sub sample to CrowdFlower for tagging.
- QA results against the baseline. If process needs improvement — do another cycle.



Web Model - Tagging

INTERFACE



Web Model - Tagging

INTERFACE

Rules and Tips

Do This

- **Browse** through at least **5 sub-pages of the website** to get a **general idea of the business**.
- **Look** for the "**About Us**" or "**Our Team**" sections of the page. Sometimes they will explicitly say what the business industry is or provide a description which is close enough.
- **Look** for **testimonials** or **reviews** posted on the **website** by the **company**. Many times the reviews will contain descriptions of the type of services that the company has provided. These services are very direct indicators of the companies industry (e.g if the services was "created a webpage" then the industry may be "Software and Web Development").

Be Careful Of

- Some companies provide multiple services that seem to match a number of industries. In such cases try to to figure out what the "**main**" **product** on line of business is and use that to decide on an industry.

Do Not

- If none of the provided industries match the given website then "other" should be chosen - **do not force an industry on a website which really doesn't match anything.**
- That said - please read carefully the description of each industry before deciding that nothing fits. Choosing "other" should be a last resort after all possible alternatives have been considered carefully.

Web Model - Tagging

INTERFACE

Step 1: Review the Company Website

Company Name	hapco
Company Website	Click Here for Company Website www.hapco.com Copy and Paste URL to double check if the website is down

Step 2: Categorize the Website

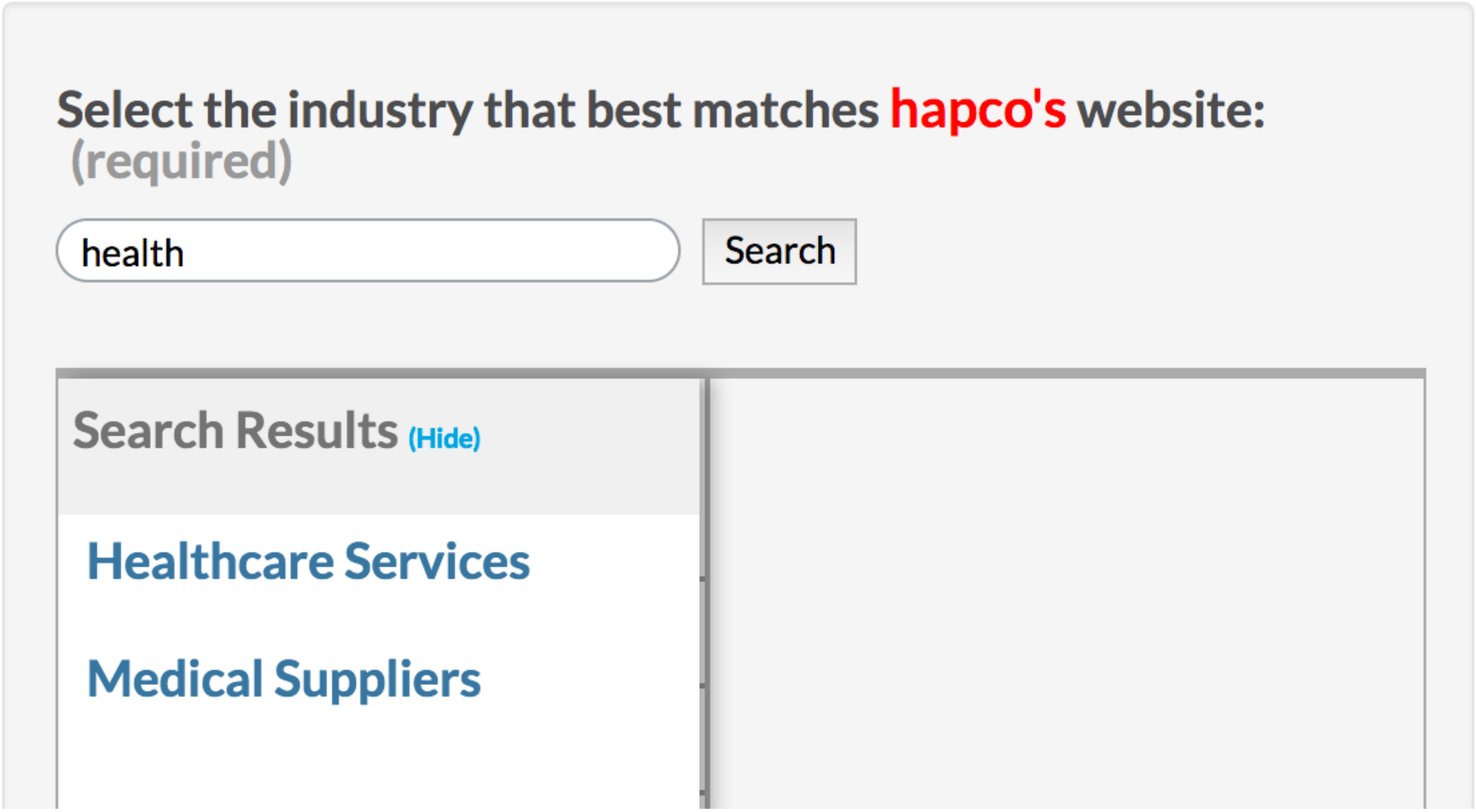
Is the website: www.hapco.com down? (required)

- (•) No
- ( ) Yes

ⓘ Click the Website button to visit the Website

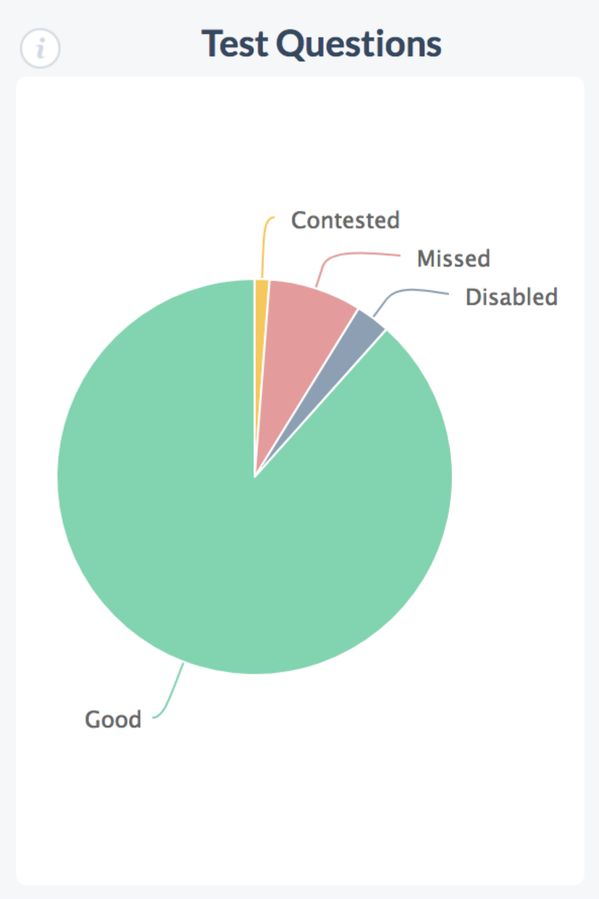
Web Model - Tagging

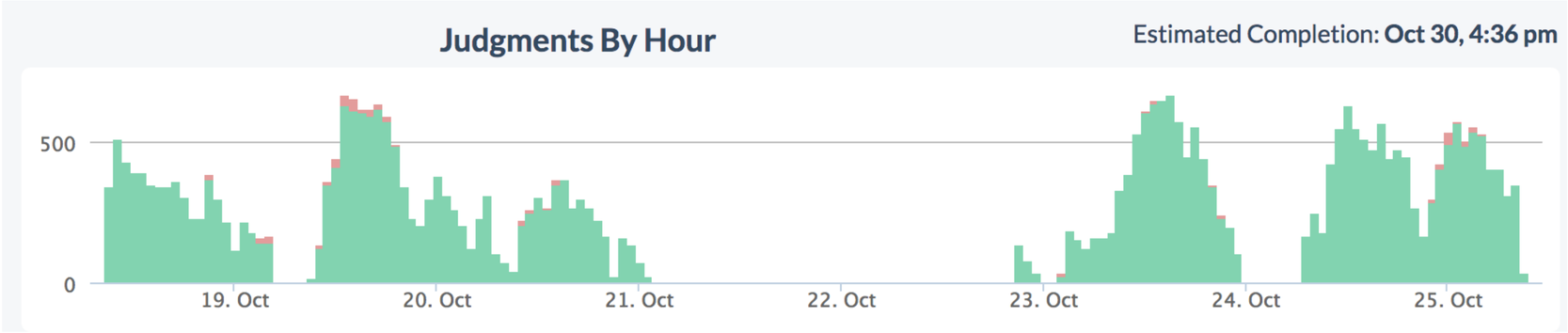
INTERFACE



Web Model - Tagging

MONITORING





Web Model - Input

THE DATASET - WEB PAGE CONTENT

BlueVine Line of Credit Invoice Factoring Why BlueVine About Us (888) 216-9619 Sign In Apply Now

Overview Overview Team In The News

BlueVine is the fastest, most convenient way for small businesses to get funding

Owning a small business can be incredibly rewarding. That's why you started yours. But it can also be challenging when cash management is taking time away from running your business and when accessing capital for your business is a slow and difficult process.

That's where we come in. We started BlueVine with two things in mind. First, to provide small businesses like yours access to capital when you need it, so you don't have to worry about making payroll or investing in growth. Second, to serve small business owners with speed, simplicity, and transparency – no long waits, paperwork, or hidden fees that happen with other financial institutions. Running a business is challenging; BlueVine makes working capital easy.

BlueVine is based in Redwood City, California and backed by leading venture capital firms and private investors. We are proud to enable small businesses to grow and thrive with our service.

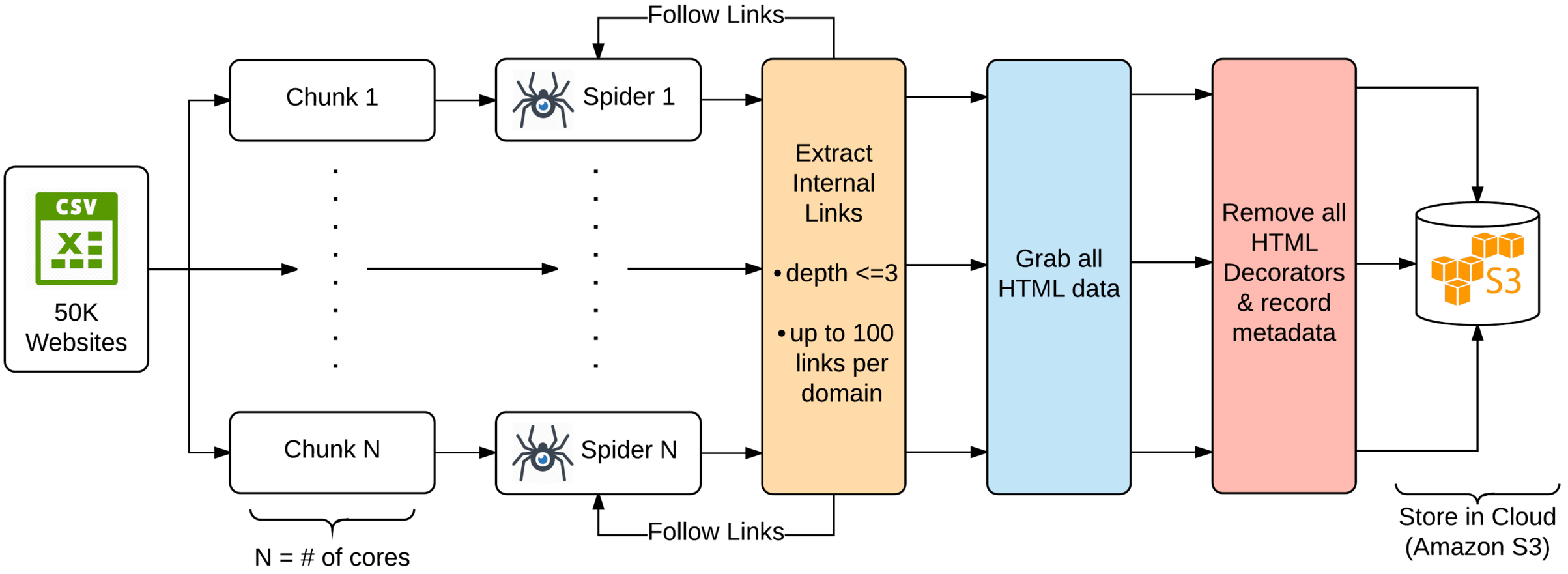
Web Model - Input

THE DATASET - WEB PAGE CONTENT


```
property="og:locale" content="en_US" /><meta
property="og:type" content="article" /><meta
property="og:title" content="About BlueVine" /><meta
property="og:description" content="Learn about BlueVine, the leadership team and recent news." /><meta
property="og:url" content="https://www.bluevine.com/about/" /><meta
property="og:site_name" content="Invoice Factoring—Fast and Simple, from BlueVine" /><meta
property="article:publisher" content="https://www.facebook.com/bluevinecapital" /><meta
property="og:image" content="https://media.bluevine.com/wp-content/uploads/2014/12/401_warren.jpg?x43706" /><meta
name="twitter:card" content="summary" /><meta
name="twitter:description" content="Learn about BlueVine, the leadership team and recent news." /><meta
name="twitter:title" content="About BlueVine" /><meta
name="twitter:site" content="@BluevineCapital" /><meta
name="twitter:image" content="https://media.bluevine.com/wp-content/uploads/2014/12/401_warren.jpg?x43706" /><meta
name="twitter:creator" content="@BluevineCapital" /><meta
property="DC.date.issued" content="2014-12-17T15:28:50+00:00" /><link
rel='dns-prefetch' href='//ajax.googleapis.com' /><link
rel='dns-prefetch' href='//s.w.org' /> <script type="text/javascript">/*<![CDATA[*/window._wpemojiSettings=
{"baseUrl":"https:\/\/s.w.org\/images\/core\/emoji\/2.3\/72x72\/","ext":".png","svgUrl":"https:\/\/s.w.org\/images\/core\/emoji\/2.3\/svg\/","svgExt":".svg","sourc
release.min.js?ver=4.8.2"}};!function(a,b,c){function d(a){var b,c,d,e,f=String.fromCharCode;if(!k||!k.fillText)return!1;switch(k.clearRect(0,0,j.width,j.height),k
k.fillText(f(55356,56826,55356,56819),0,0),b=j.toDataURL(),k.clearRect(0,0,j.width,j.height),k.fillText(f(55356,56826,8203,55356,56819),0,0),c=j.toDataURL(),b!==c&
(k.clearRect(0,0,j.width,j.height),k.fillText(f(55356,57332,56128,56423,56128,56418,56128,56421,56128,56430,56128,56423,56128,56447),0,0),b=j.toDataURL(),k.clearRe
,56418,8203,56128,56421,8203,56128,56430,8203,56128,56423,8203,56128,56447),0,0),c=j.toDataURL(),b!==c);case"emoji4":return
k.fillText(f(55358,56794,8205,9794,65039),0,0),d=j.toDataURL(),k.clearRect(0,0,j.width,j.height),k.fillText(f(55358,56794,8203,9794,65039),0,0),e=j.toDataURL(),d!=
c=b.createElement("script");c.src=a,c.defer=c.type="text/javascript",b.getElementsByTagName("head")[0].appendChild(c)}var f,g,h,i,j=b.createElement("canvas"),k=j.g
{everything:!0,everythingExceptFlag:!0},h=0;h<i.length;h++)c.supports[i[h]]=d(i[h]),c.supports.everything=c.supports.everything&&c.supports[i[h]],"flag"!==i[h]&&
(c.supports.everythingExceptFlag=c.supports.everythingExceptFlag&&c.supports[i[h]]);c.supports.everythingExceptFlag=c.supports.everythingExceptFlag&&!c.supports.fl
{c.DOMReady=!0},c.supports.everything||(g=function(){c.readyCallback()},b.addEventListener?(b.addEventListener("DOMContentLoaded",g,!1),a.addEventListener("load",g
(a.attachEvent("onload",g),b.attachEvent("onreadystatechange",function(){"complete"===b.readyState&&c.readyCallback()})),f=c.source||{},f.concatemoji?e(f.concatemo
(window,document,window._wpemojiSettings);/*]]>*/</script> <style type="text/css">img.wp-smiley,img.emoji{display:inline !important;border:none !important;box-shad
!important;vertical-align:-0.1em !important;background:none !important;padding:0 !important}</style> <script type="text/javascript" src="https://media.bluevine.com
type='text/javascript'>var notices_ajax_script={"ajaxurl":"https:\/\/www.bluevine.com\/wp-admin\/admin-ajax.php"};</script> <script type="text/javascript" src="htt
</script> <script type="text/javascript">/*<![CDATA[*/!function(){var analytics=window.analytics=window.analytics||[];if(analytics.invoked)window.console&&console.
twice.");else{analytics.invoked=!0;analytics.methods=["trackSubmit","trackClick","trackLink","trackForm","pageview","identify","group","track","ready","alias","pag
e=Array.prototype.slice.call(arguments);e.unshift(t);analytics.push(e);return analytics}};for(var t=0;t<analytics.methods.length;t++){var e=analytics.methods[t];an
```

Web Model - Scraping

BASIC FLOW



Web Model - Scraping

IMPLEMENTATION

- Built using Scrapy library.

- Need to define:
  - Spiders — which sites to crawl & some settings (e.g. wether to follow links).
  - Pipelines — what to do with the scraped data (processing).
  - Exporters — where & how to save the output.
  - Settings / Middleware — tweaks.

- Everything else is handled automatically.

Web Model - Scraping

IMPLEMENTATION

Scrapy Input

url_id	url
1	www.amazon.com
2	www.odsc.com
...	...
50,000	www.pizzahut.com

Scrapy Output

url_id	link_number	url	error_code	page_data
1	1	www.amazon.com		<data>
1	2	www.amazon.com/tools		<data>
1	3	www.amazon.com/books		<data>
2	1	www.odsc.com	404	<EMPTY>
...	...	...		...
50,000	1	www.pizzahut.com		...
50,000	2	www.pizzahut.com/pizzas		...
50,000	3	www.pizzahut.com/toppings		...
50,000	4	www.pizzahut.com/prices		...

Web Model - Scraping

IMPLEMENTATION - SPIDERS [SCRAPY] - Response “Meta”

```
class IndustrySpider(CrawlSpider):
    name = "industry_spider"
    start_urls = []  # LIST OF URLS HERE
    start_url_ids = [] # LIST OF URL IDS HERE

    def start_requests(self):
        for url, url_id in zip(self.start_urls, self.start_url_ids):
            yield self.custom_request(url, url_id, link_position=1, callback=self.parse_first_url)


    def custom_request(self, url, url_id, link_position, callback):
        Request(url=url,

                callback=callback,

                meta={'url_id': url_id,
                      'link_position': link_position},

                errback=lambda x: self.error_handler(x, {'url': url,
                                                         'url_id': url_id,
                                                         'link_position': link_position}))
```

Web Model - Scraping

IMPLEMENTATION - SPIDERS [SCRAPY] - Response “Meta”

```
class IndustrySpider(CrawlSpider):

    ...

    def parse_first_url(self, response):
        url_id = response['meta'].url_id
        url_domain = urlparse(response).netloc
        internal_links = enumerate(LinkExtractor(allow_domains=[url_domain]).extract_links(response))
        for output in [self.parse_link(response)] + [self.custom_request(link.url, url_id, link_position + 1, callback=self.parse)
                                                     for (link, link_position) in internal_links]
            yield output

    def parse(self, response):
        web_page = WebPage()
        web_page['url'] = response.url
        web_page['url_id'] = response.meta['url_id']
        web_page['link_number'] = response.meta['link_number']
        try:
            web_page['page_data'] = response.body.decode("utf-8")
        except UnicodeDecodeError:
            web_page['page_data'] = response.body.decode("ISO-8859-1")
        return web_page
```

Web Model - Scraping

IMPLEMENTATION - SPIDERS [SCRAPY] - Response “Meta”

```
class IndustrySpider(CrawlSpider):

    ...

    def error_handler(self, failiure, meta):
        web_page = WebPage()
        web_page['url'] = response.url
        web_page['url_id'] = response.meta['url_id']
        web_page['link_number'] = response.meta['link_number']
        web_page['err_message'] = str(failiure.getErrorMessage())
        return web_page
```

Web Model - Scraping

IMPLEMENTATION - SPIDERS [SCRAPY] - Response “Meta”

```
class WebPageCleanerPipeline:

    ...

    def process_item(self, item, spider):
        item['page_data_clean'] = clean_web_page(item['page_data'])  # REMOVE HTML AND OTHER WEIRD THINGS

        raw_data_item = {k: v for (k, v) in dict(item).items() if k != 'page_data_clean'}
        self.raw_data_all = self.raw_data_all.append(pd.DataFrame([clean_data_item]))

        clean_data_item = {k: v for (k, v) in dict(item).items() if k == 'page_data_clean'}
        self.clean_data_all = self.clean_data_all.append(pd.DataFrame([clean_data_item]))

        self.item_counter += 1
        if self.item_counter % self.upload_step == 0:  # UPLOAD DATA TO CLOUD AT INCREMENTS
            ...
```

Web Model - Scraping

IMPLEMENTATION - SPIDERS [SCRAPY] - MIDDLEWARE/SETTINGS

```
BOT_NAME = 'industry_crawler'
SPIDER_MODULES = ['industry_crawler.spiders']
NEWSPIDER_MODULE = 'industry_crawler.spiders'
EXTENSIONS = {
    'scrapy.extensions.closespider.CloseSpider': 500
}
COOKIES_ENABLED = False
AJAXCRAWL_ENABLED = True
LOG_LEVEL = 'INFO'
REDIRECT_ENABLED=True
ROBOTSTXT_ENABLED=False
DEPTH_LIMIT = 3
DOWNLOAD_DELAY = 0.5
DOWNLOAD_TIMEOUT = 20
```

```
ITEM_PIPELINES = {
    'industry_crawler.pipelines.WebPageCleanerPipeline': 300,
}
```

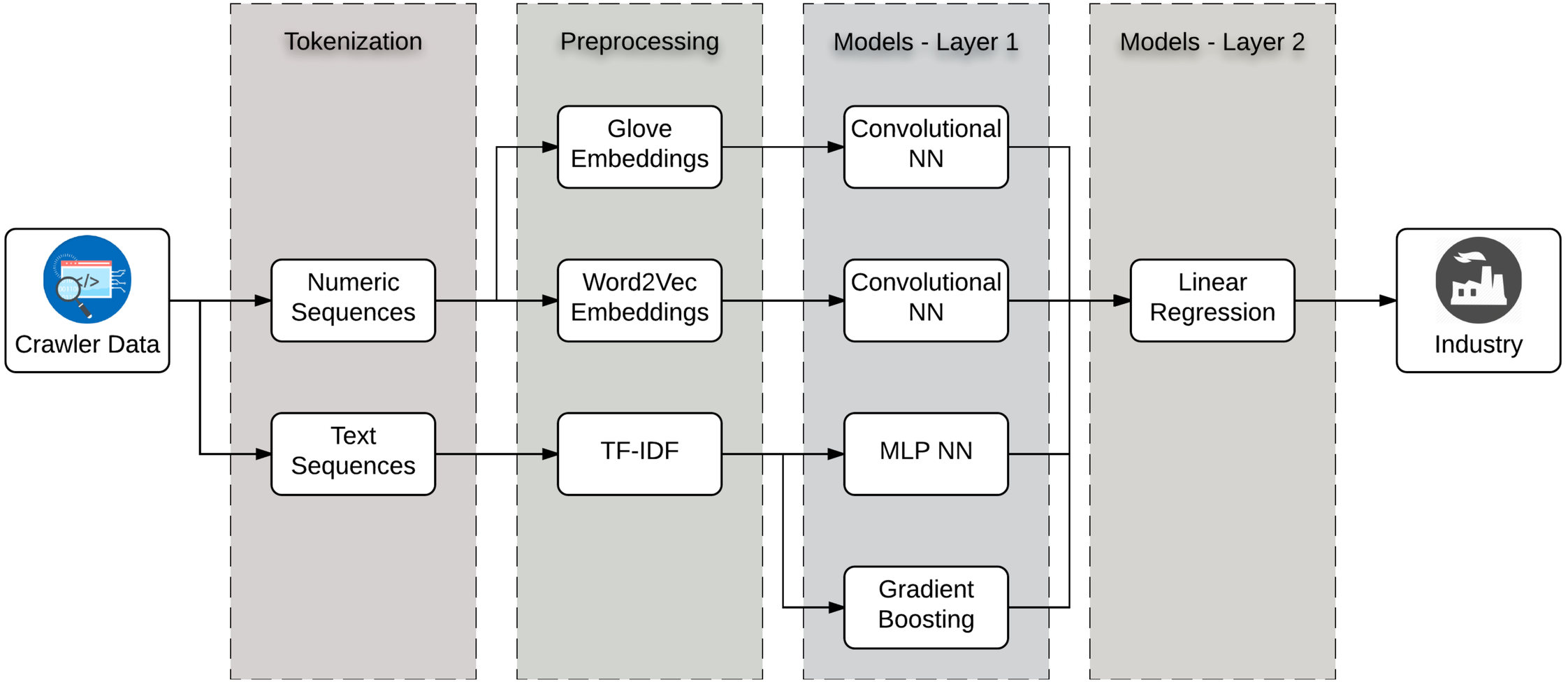
```
DOWNLOADER_MIDDLEWARES = {
    'industry_crawler.middlewares.FilterDomainbyLimitMiddleware': 543,
}
```

```
MAX_REQUESTS_PER_DOMAIN = 100
```

```
class FilterDomainbyLimitMiddleware(object):
    def __init__(self, max_requests_per_domain):
        self.max_requests_per_domain = max_requests_per_domain
        self.counter = defaultdict(int)

    @classmethod
    def from_crawler(cls, crawler):
        settings = crawler.settings
        max_requests_per_domain = \
            settings.get('MAX_REQUESTS_PER_DOMAIN')
        o = cls(max_requests_per_domain)
        return o

    def process_request(self, request, spider):
        parsed_url = urlparse(request.url)
        if self.counter.get(parsed_url.netloc, 0) < \
           self.max_requests_per_domain:
            self.counter[parsed_url.netloc] += 1
        else:
            raise IgnoreRequest()
```

Web Model - Design

HIGH LEVEL FLOW



Web Model - Design

TOKENIZATION

- Split text into pieces, or “tokens”. Use them to represent the text.

- Used as stepping stone represent text in a numeric way.

- Common practices:
  - Lowercase everything
  - Remove punctuation
  - Limit vocabulary (e.g. keep only the most frequent N tokens)
  - Remove inflections (stemming / lemmatization)

$$\text{am, are, is} \Rightarrow \text{be}$$
$$\text{car, cars, car's, cars'} \Rightarrow \text{car}$$

Web Model - Design

T O K E N I Z A T I O N

The problem - **CANNOT be parallelized!** (at least not trivially…)

Keras “Tokenizer”

- Converts texts into sequences of integers
- Each token has a **UNIQUE NUMBER**

Sklearn “TfidfVectorizer”

- Converts texts into numeric vectors
- Each vector contains **ALL TOKENS** in the corpus

Web Model - Design

TOKENIZATION - IN PARALLEL! [SPACY]

```
from joblib import cpu_count, Parallel, delayed
import math

# SPLIT CORPUS INTO CHUNKS
num_jobs_per_chunk = 5
num_chunks = int(math.floor(cpu_count() / num_jobs_per_chunk))
corpus_chunks = split_into_k_chunks(corpus, num_chunks)

# DEFINE FUNCTION TO HANDLE CHUNK TOKENIZATION
def tokenize_via_spacy(*args, **kwargs):
    ...

# RUN TOKENIZATION ON CHUNKS IN PARALLEL
Parallel(n_jobs=num_chunks, backend='multiprocessing')(
    delayed(tokenize_via_spacy)(chunk_id, corpus_chunk, num_jobs_per_chunk, word2vec_model, glove_model)
    for chunk_id, corpus_chunk in enumerate(corpus_chunks))
```

Web Model - Design

TOKENIZATION - IN PARALLEL! [SPACY]

```
def tokenize_via_spacy(chunk_id, corpus_chunk, num_jobs_per_chunk):

    nlp = spacy.load('en')  # LOAD SPACY ENGLISH MODEL

    word_counts = {}        # RECORD FREQUENCY OF EVERY TOKEN
    word_indices = []       # GENERATE STREAM OF "TOKEN NUMBERS"
    word_texts = []         # GENERATE STREAM OF "TOKEN TEXTS"

    nlp_pipe = nlp.pipe(corpus_chunk, batch_size=100, n_threads=num_jobs_per_chunk)
    for doc in nlp_pipe:
        doc_word_indices = []
        doc_word_texts = []
        for word in doc:
            if word.is_stop:
                continue
            word_counts[word.lemma] += 1 if word.lemma in word_counts.keys()
                else word_counts[word.lemma] = 1
            doc_word_indices += [word.lemma]
            doc_word_texts += [word.lemma_]
        word_indices += [doc_word_indices]
        word_texts += [doc_word_texts]

    save_everything_to_disk(word_counts, word_indices, word_texts)
```

word = "cats"

word.lemma_ = "cat"

word.lemma = 19247

Web Model - Design

TOKENIZATION - IN PARALLEL! [SPACY]

```
# LOAD AND UNIFY DATA FROM ALL CHUNKS
word_counts = []
word_indices = []
word_texts = []

for chunk_id, _ in enumerate(corpus_chunks):
    batch_word_counts, chunk_word_indices, chunk_word_texts = load_everything_from_disk(file_paths)
    word_counts += [chunk_word_counts]
    word_indices += [chunk_word_indices]
    word_texts += [word_texts]

word_counts = sum_dictionaries(word_counts)
word_indices = combine_dictionaries(word_indices)
word_texts = combine_dictionaries(word_texts)
```

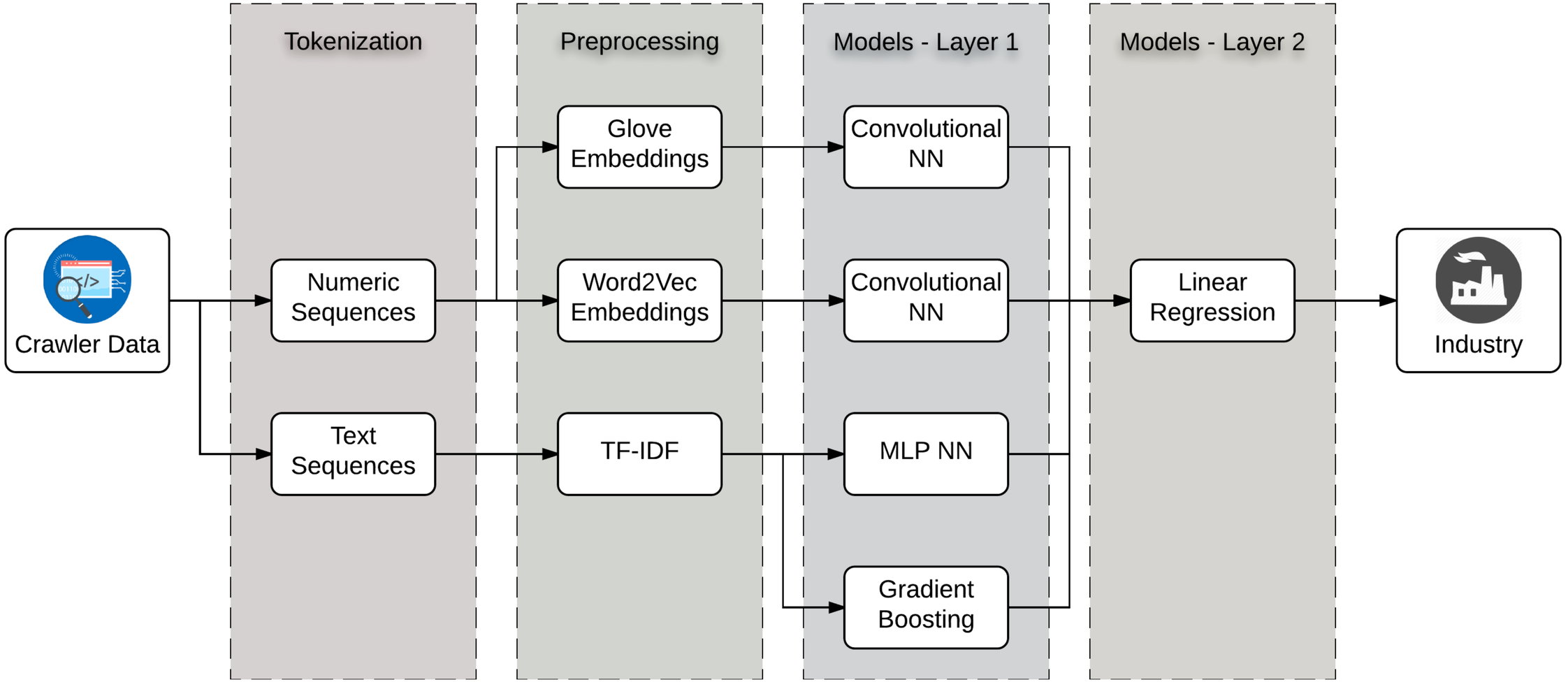
Web Model - Design

TOKENIZATION - IN PARALLEL! [SPACY]

```
# GET TOP N WORDS
word_counts_sorted = sorted(word_counts.items(), key=operator.itemgetter(1), reverse=True)
top_words = [word for (word, word_count) in word_counts_sorted[:N]]


# RESET WORD INDICES FROM 1 TO N
index_transform = dict(zip(top_words, list(range(1, len(top_words) + 1))))
word_sequences_transformed = [[index_transform[word] for word in sequence]
                              for sequence in word_indices]

# GET TF-IDF WEIGHTS
tf_idf = TfidfVectorizer(ngram_range=(1, 2),
                         lowercase=False,
                         preprocessor=lambda x: x,
                         tokenizer=lambda x: x)
tf_idf_matrix = tf_idf.fit_transform(word_texts)

reduced_dimension = int(round(np.sqrt(tf_mat.shape[0]), 0))
svd = TruncatedSVD(reduced_dimension)
tf_idf_reduced_matrix = clf.fit_transform(tf_idf_matrix)
```

Web Model - Design

HIGH LEVEL FLOW



Web Model - Design

EMBEDDINGS - BACKGROUND

- A word is represented as a **VECTOR** of size 1XN.

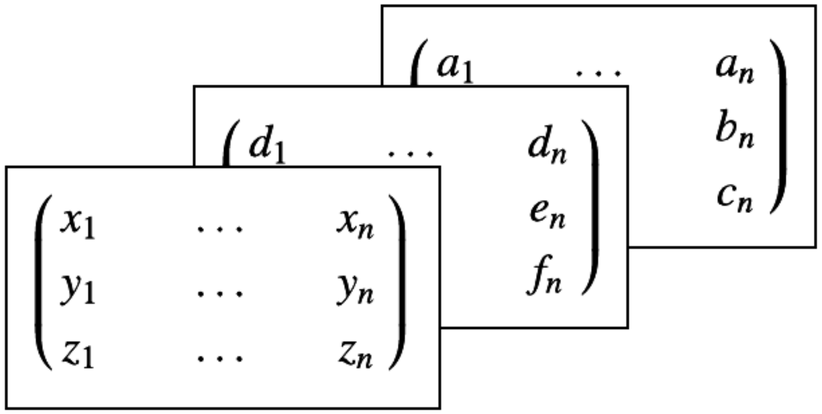
$$\begin{aligned} \text{"Quick"} &= (x_1, \ldots, x_n) \\ \text{"Brown"} &= (y_1, \ldots, y_n) \\ \text{"Fox"} &= (z_1, \ldots, z_n) \end{aligned}$$

- A string with K words is represented as a **MATRIX** of size KXN.

$$\text{"Quick Brown Fox"} = \begin{pmatrix} x_1 & \ldots & x_n \\ y_1 & \ldots & y_n \\ z_1 & \ldots & z_n \end{pmatrix}$$

- A corpus (collection) of M strings is represented as a **TENSOR** of size MXKXN.

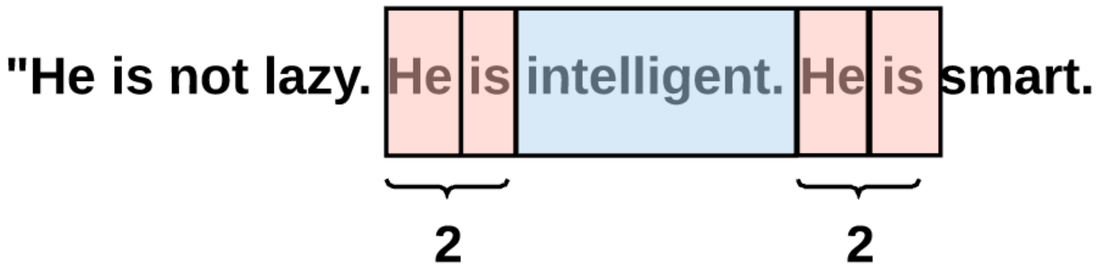
$$\begin{pmatrix} \text{"Quick Brown Fox"} \\ \text{"Jumped Over The"} \\ \text{"Lazy Brown Dog"} \end{pmatrix} =$$



Web Model - Design

EMBEDDINGS - CONCURRENCY

- Every word is represented by the words that frequently appear with it.
- For a string with N words, you get an NXN matrix of representations called a **concurrency matrix**.
- The word vectors are the rows / columns of the matrix.



	he	is	not	lazy	intelligent	smart
he	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

Web Model - Design

EMBEDDINGS - CONCURRENCY

- Number of words in a corpus = dimension of embedding (word vector length).
- A corpus can easily have hundreds of thousands of unique words / terms.
- Most models can’t use so many features. Dimensionality reduction is applied to make the embeddings usable without losing (too much) information (e.g. PCA).

$$PCA: \underbrace{\begin{pmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \dots & x_{n,n} \end{pmatrix}}_{\text{Original Embedding obs=n, dim=n (nXn)}} = \begin{pmatrix} u_{1,1} & \dots & u_{1,r} \\ \vdots & \ddots & \vdots \\ u_{N,1} & \dots & u_{b,r} \end{pmatrix} \cdot \begin{pmatrix} s_{1,1} & 0 & \dots \\ 0 & \ddots & \dots \\ \vdots & \dots & s_{r,r} \end{pmatrix} \cdot \underbrace{\begin{pmatrix} v_{1,1} & \dots & v_{1,n} \\ \vdots & \ddots & \vdots \\ v_{r,1} & \dots & v_{n,n} \end{pmatrix}}_{\text{Reduced Embedding obs=n, dim=r (rXn)}}$$

Web Model - Design

EMBEDDINGS - GLOVE

- Glove embeddings: generated from a concurrency matrix trained on Wikipedia, Common Crawl or Twitter data (billions of words).

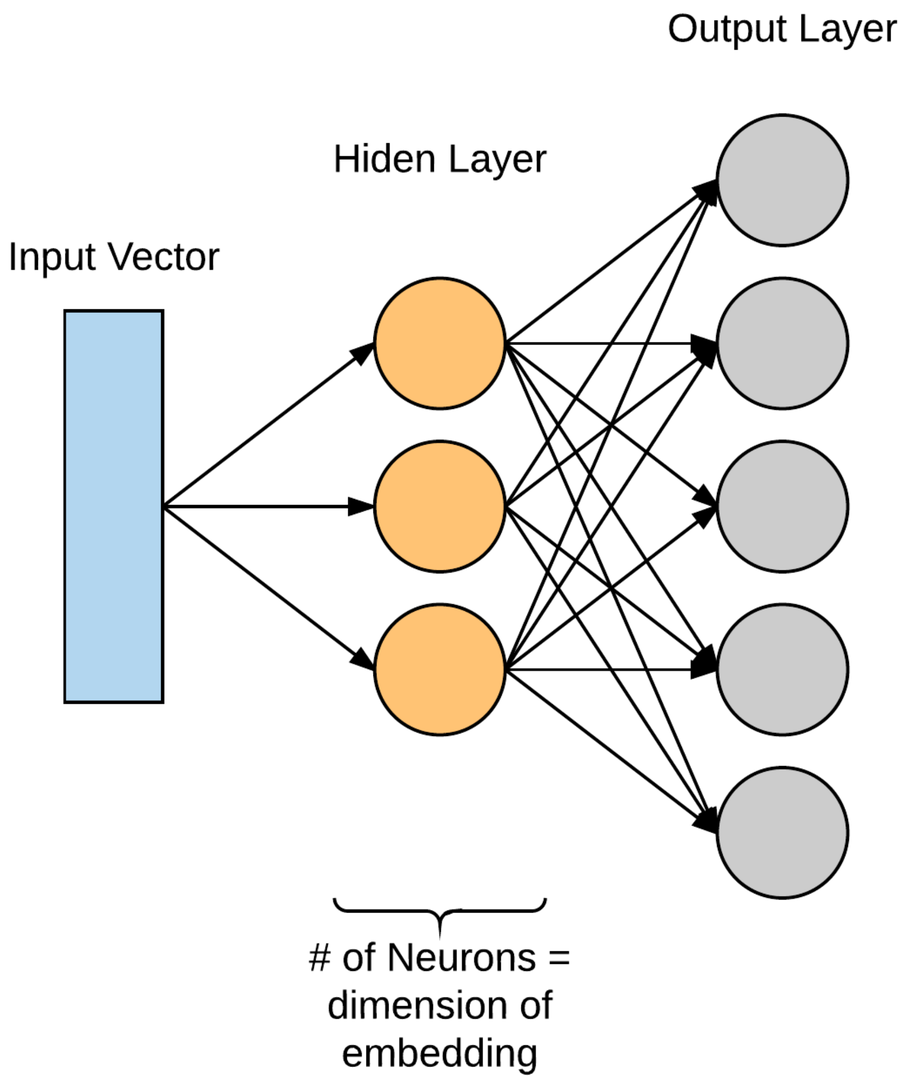
- With reduced dimensions: 100, 200, 300 or 400 (depending on source).

- Can be downloaded and inserted into models as layers.

Web Model - Design

EMBEDDINGS - SKIPGRAM

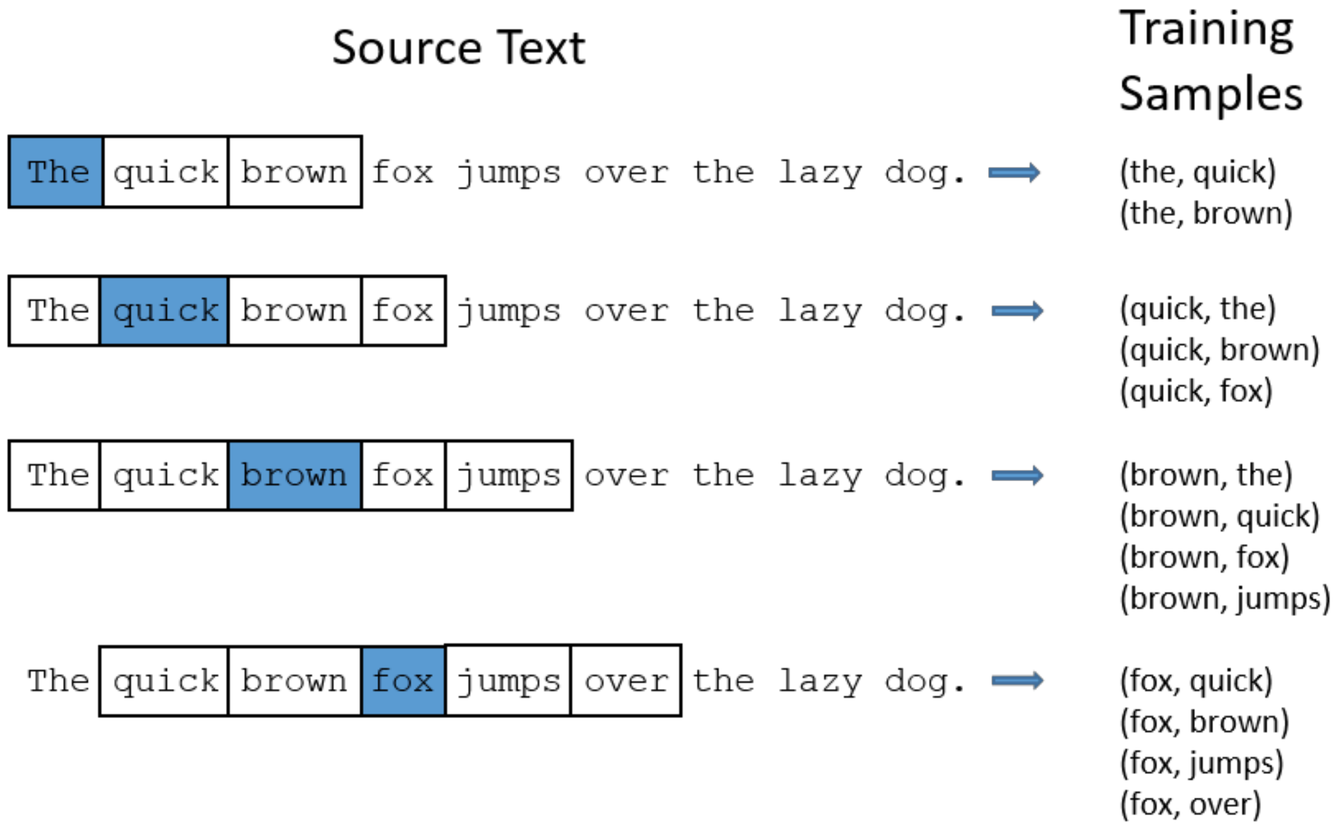
- Every word is represented by the vector of weights for a hidden layer in a trained neural network.
- We **DON'T** care about the final prediction of the network. We **DO CARE** about the final weights of the hidden layer.
- The type of model used to generate the weight vectors is called Skip Gram.



Web Model - Design

EMBEDDINGS - SKIPGRAM

- Network is trained on word pairs, an “input” word and a “target” word.
- Set a window size (say 2) around an “input” word. All other words inside that window are “targets” for that word.



Web Model - Design

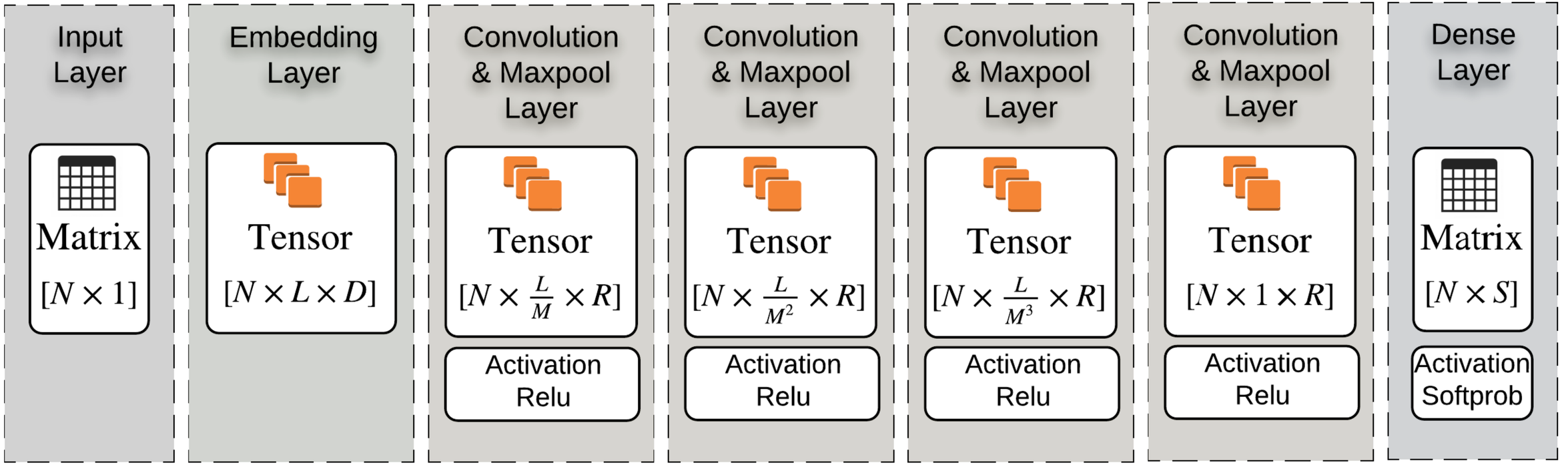
EMBEDDINGS - WORD 2 VEC

Pre-trained Word 2 Vec embeddings are

- Trained by Google on Google News dataset (about 100 billion words).
- Have a dimension of 300.
- Can be downloaded and inserted into models as layers.

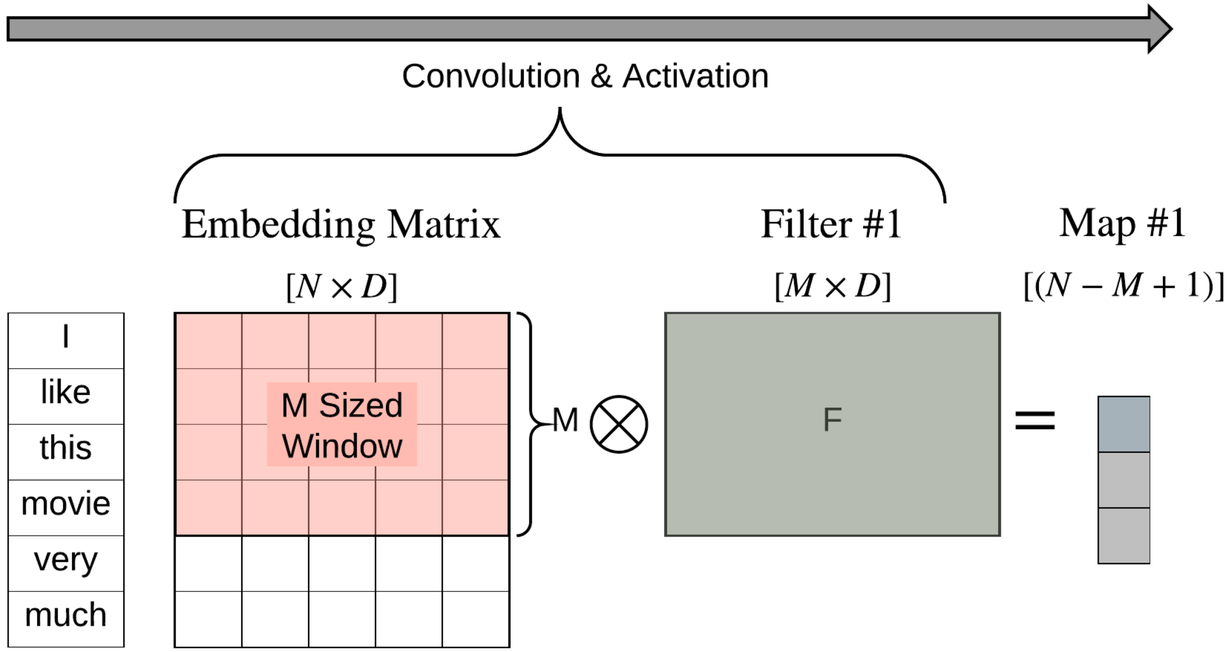
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



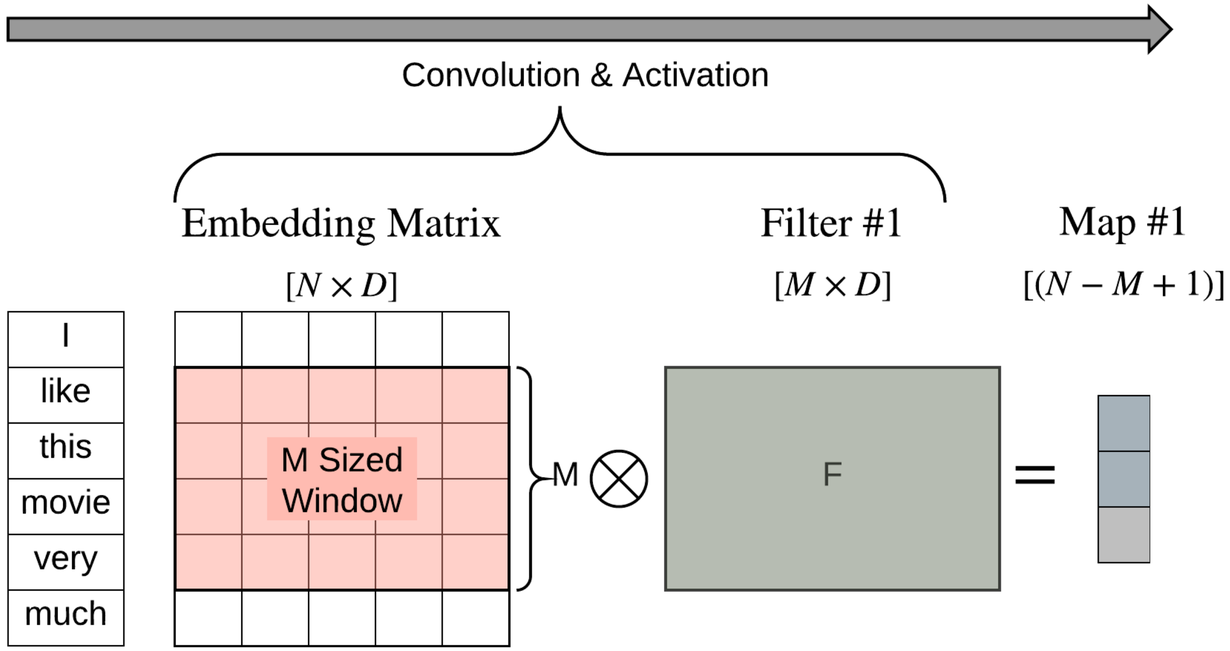
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



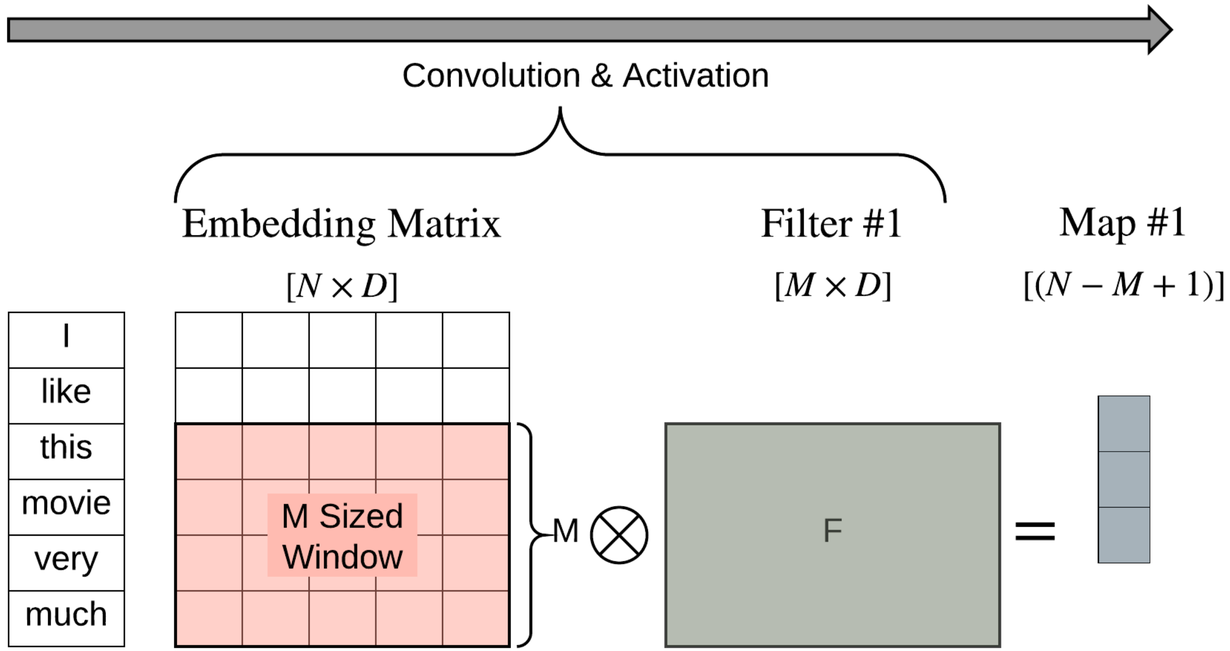
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



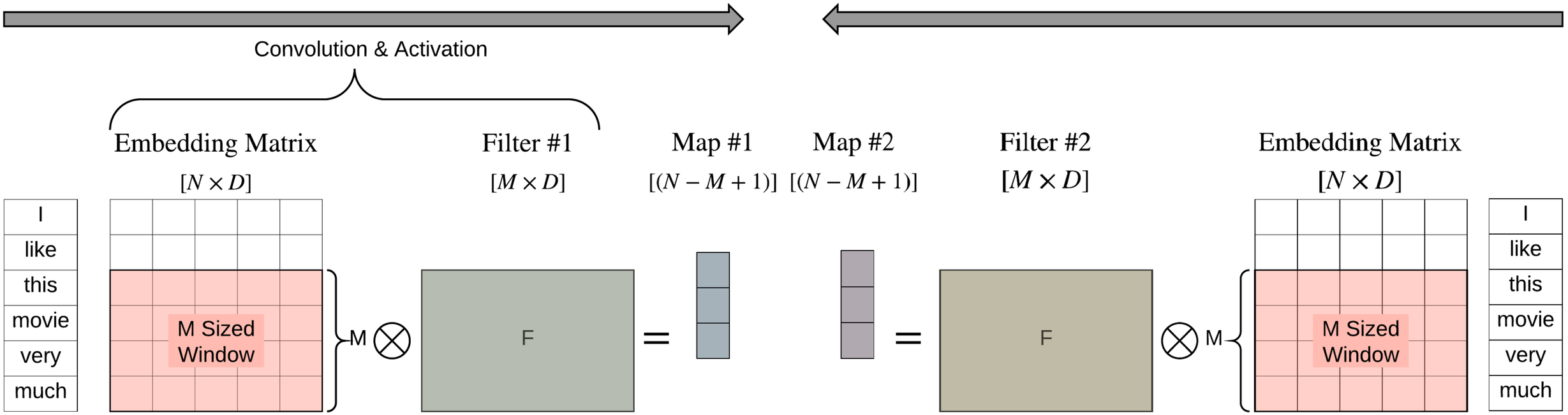
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



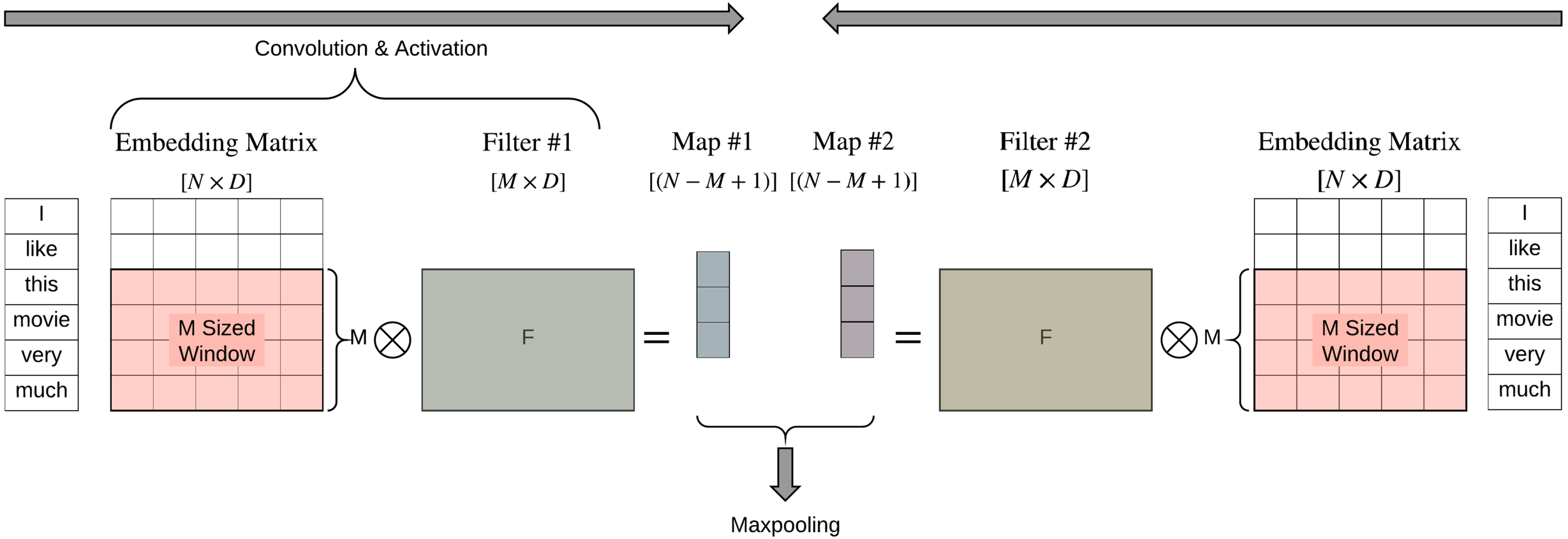
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



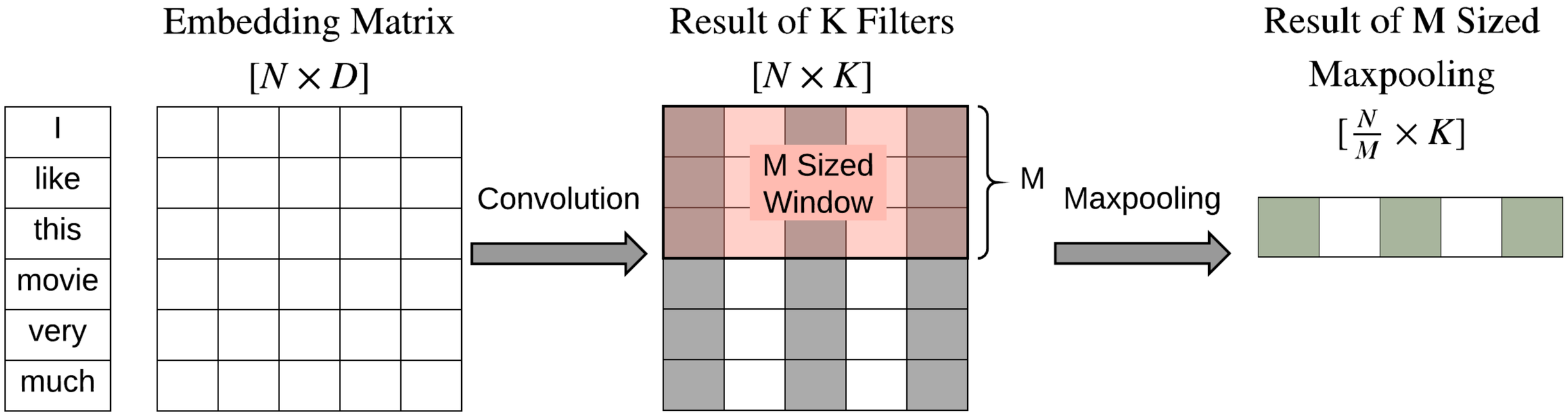
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



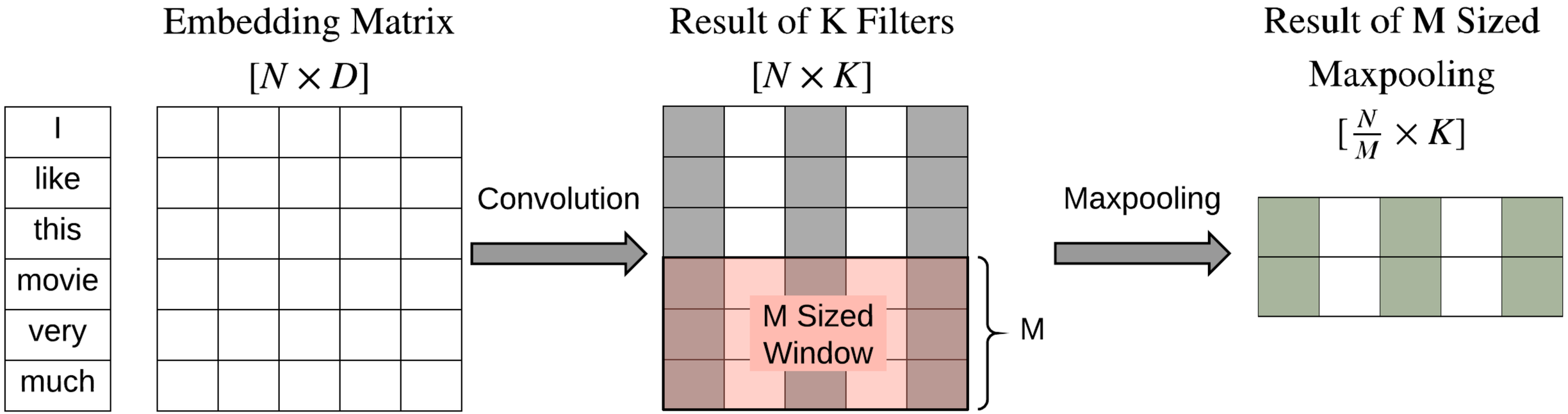
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN



Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN

```
import numpy as np
from gensim.models.keyedvectors import KeyedVectors


# LOAD EMBEDDING MODELS (GLOVE & WORD 2 VEC)
word2vec_model = KeyedVectors.load_word2vec_format(word2vec_embeddings_fp, binary=True)
glove_model = load_glove_embeddings_into_dict(glove_embedding_fp)

# GENERATE EMBEDDING MATRICES
word2vec_word_index = [word2vec_model[word] for word in top_words]
glove_word_index = [glove_model[word] for word in top_words]

word2vec_embedding_matrix = np.matrix(word2vec_word_index, max_sequence_length)
glove_embedding_matrix = np.matrix(glove_word_index, max_sequence_length)
```

Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN [KERAS]

```
X
```

$$\begin{pmatrix} [1,4,5,3] \\ [2,5,1,6,1] \\ [7,8] \end{pmatrix}$$

```
X = pad_sequences(X, maxlen=max_sequence_length)
```

$$\begin{pmatrix} 0 & 1 & 4 & 5 & 3 \\ 2 & 5 & 1 & 6 & 1 \\ 0 & 0 & 0 & 7 & 8 \end{pmatrix}$$

```
embedding_matrix
```

$$\begin{pmatrix} [\text{embedding vector for: } 1] \\ [\text{embedding vector for: } 2] \\ \vdots \\ [\text{embedding vector for: } N] \end{pmatrix}$$

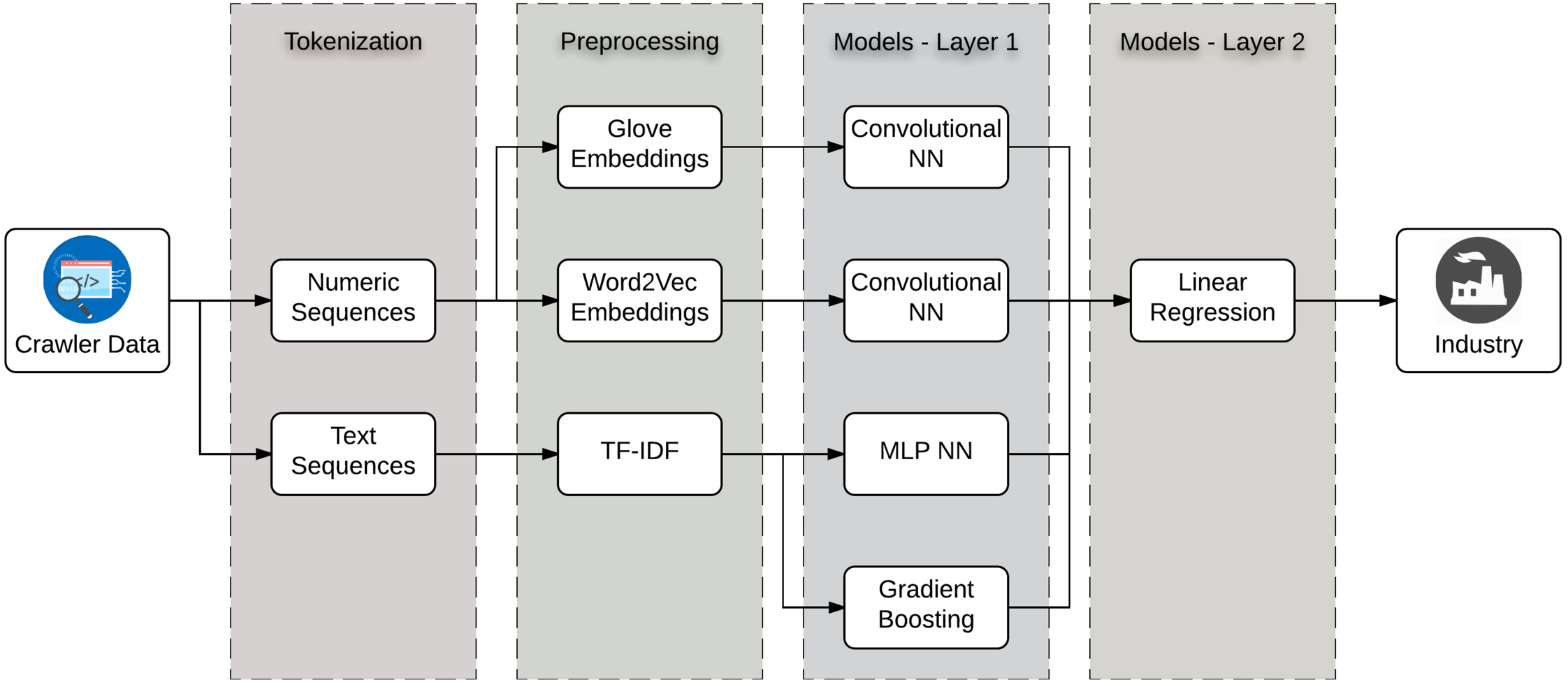
Web Model - Design

EMBEDDING MODELS - CONVOLUTIONAL NN [KERAS]

```
input_layer = Input(shape=(max_sequence_length,), dtype='int32')
x = Embedding(data_input_dim, embedding_dim, weights=[embedding_matrix],
              input_length=max_sequence_length, trainable=False)(input_layer)
x = Conv1D(batch_size, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(batch_size, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(batch_size, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(batch_size, 5, activation='relu')(x)
x = MaxPooling1D(75)(x)
x = Flatten()(x)
x = Dense(batch_size, activation='relu')(x)
out = Dense(num_pred_categories, activation='softmax')(x)

model = Model(input_layer, out)
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])
```

Web Model - Design

HIGH LEVEL FLOW



Web Model - Design

TF-IDF

- A string is represented as a vector of size 1XN, where N = number of tokens in corpus.
- Each token is given a numeric value, or a “weight”.
- “TF” is **HIGH** if a term (token) is frequent in a specific string.
- “IDF” is **HIGH** if a term (token) is rare in the entire document.

$$\text{weight}(w) = \underbrace{\text{tf}(w)}_{\text{Term Frequency}} \times \underbrace{\text{idf}(w)}_{\text{Inverse Document Frequencey}}$$

Web Model - Design

TF-IDF

corpus = "the sky is blue"
"the sky is good"
"candy is candy"

	is	sky	the	candy	blue	good
s1 freq	1	1	1	0	1	0
s2 freq	1	1	1	0	0	1
s3 freq	1	0	0	2	0	0
total freq	3	2	2	2	1	1
num docs	3	2	2	1	1	1

	is	sky	the	candy	blue	good
s1 vec	1	1.29	1.29	0.00	1.69	0.00
s2 vec	1	1.29	1.29	0.00	0.00	1.69
s3 vec	1	0.00	0.00	3.39	0.00	0.00

candy

- unique to only 1 string
- very frequent inside that 1 string
- **high** value!

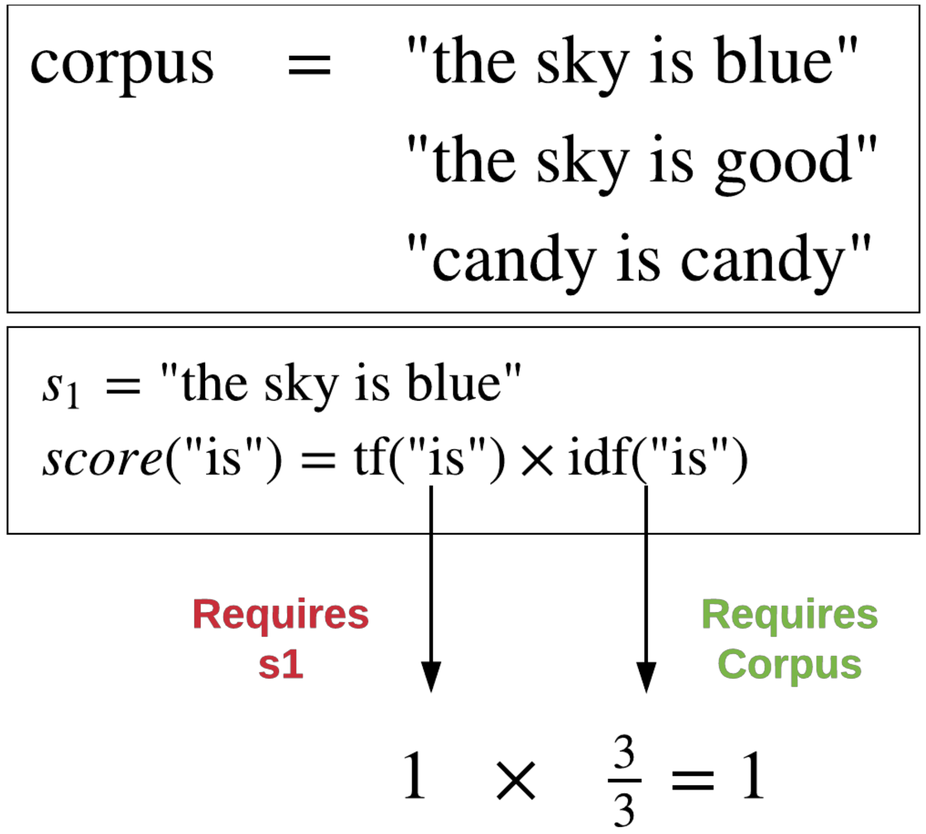
is

- appears in all strings.
- not frequent inside the strings.
- **low** value!

Web Model - Design

TF-IDF - HASHING TRICK

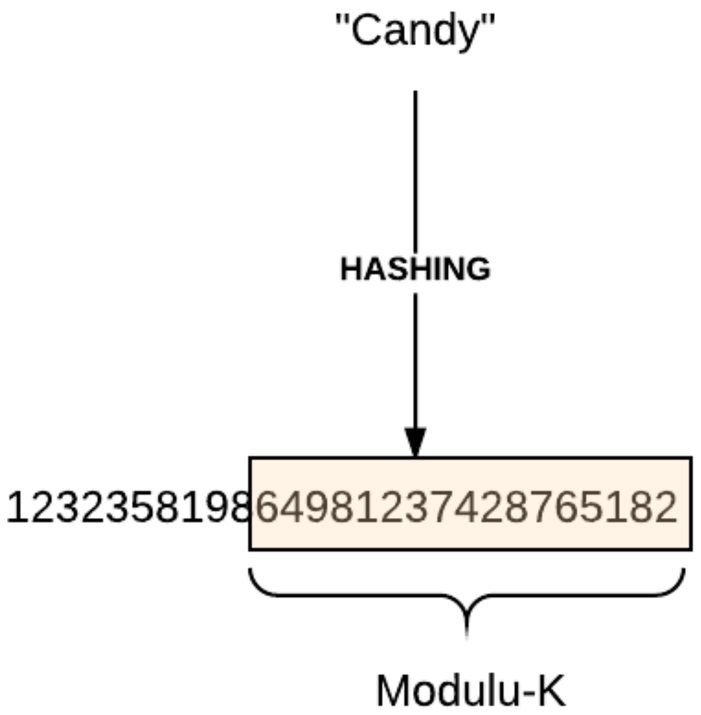
- Training a TF-IDF vectorizer is time consuming and memory consuming.
- The process **CAN'T** be parallelized since it is "stateful". Each word is weighted compared to the entire corpus, so corpus can't be split into independent pieces.
- This creates a significant bottleneck.



Web Model - Design

TF-IDF - HASHING TRICK

- Step 1: Set a desired dimension K. Final transformed output will be of size KXK.
- Step 2:Tokenize words using a hashing function. Each word is mapped to an N-digit number, the last K<N digits are chosen.
- Step 3: Apply TF-IDF transformation on tokenized matrix. This is very fast since its just basic matrix arithmetic.



Web Model - Design

TF-IDF - HASHING TRICK

Pros

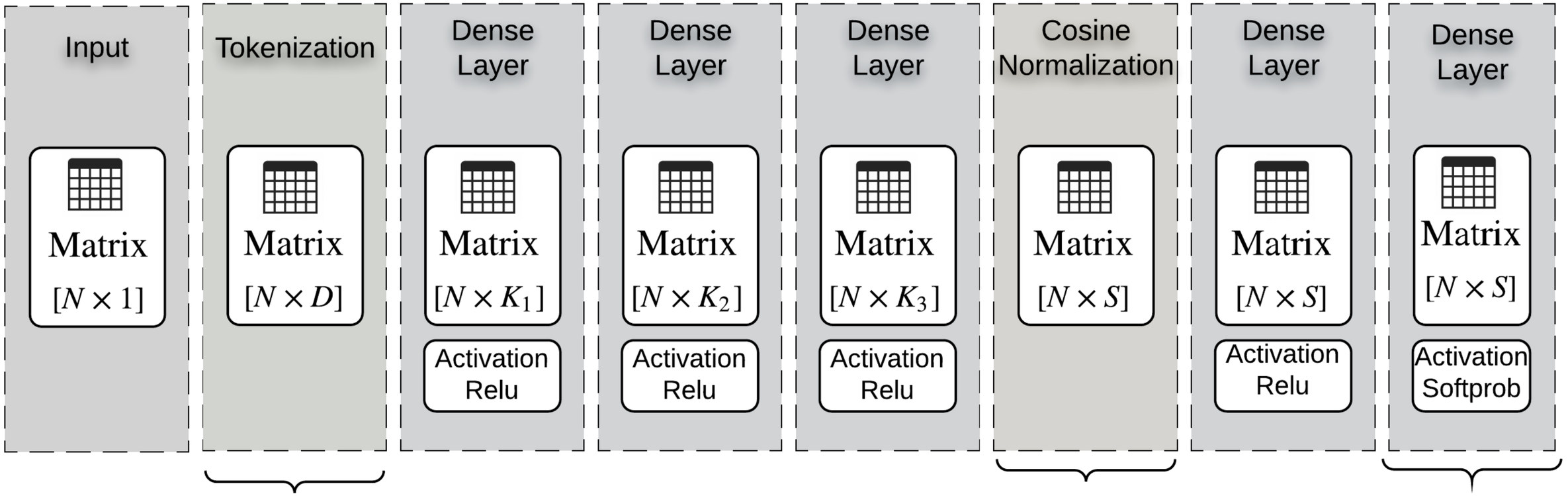
- Tokenization is stateless, so the process can be parallelized.
- No dictionary to store, so low use of memory.

Cons

- No (easy) way to map tokens back into words.
- Possibility of collisions.
- Dimension may be unnecessarily high

Web Model - Design

TF-IDF MODELS - MLP NN



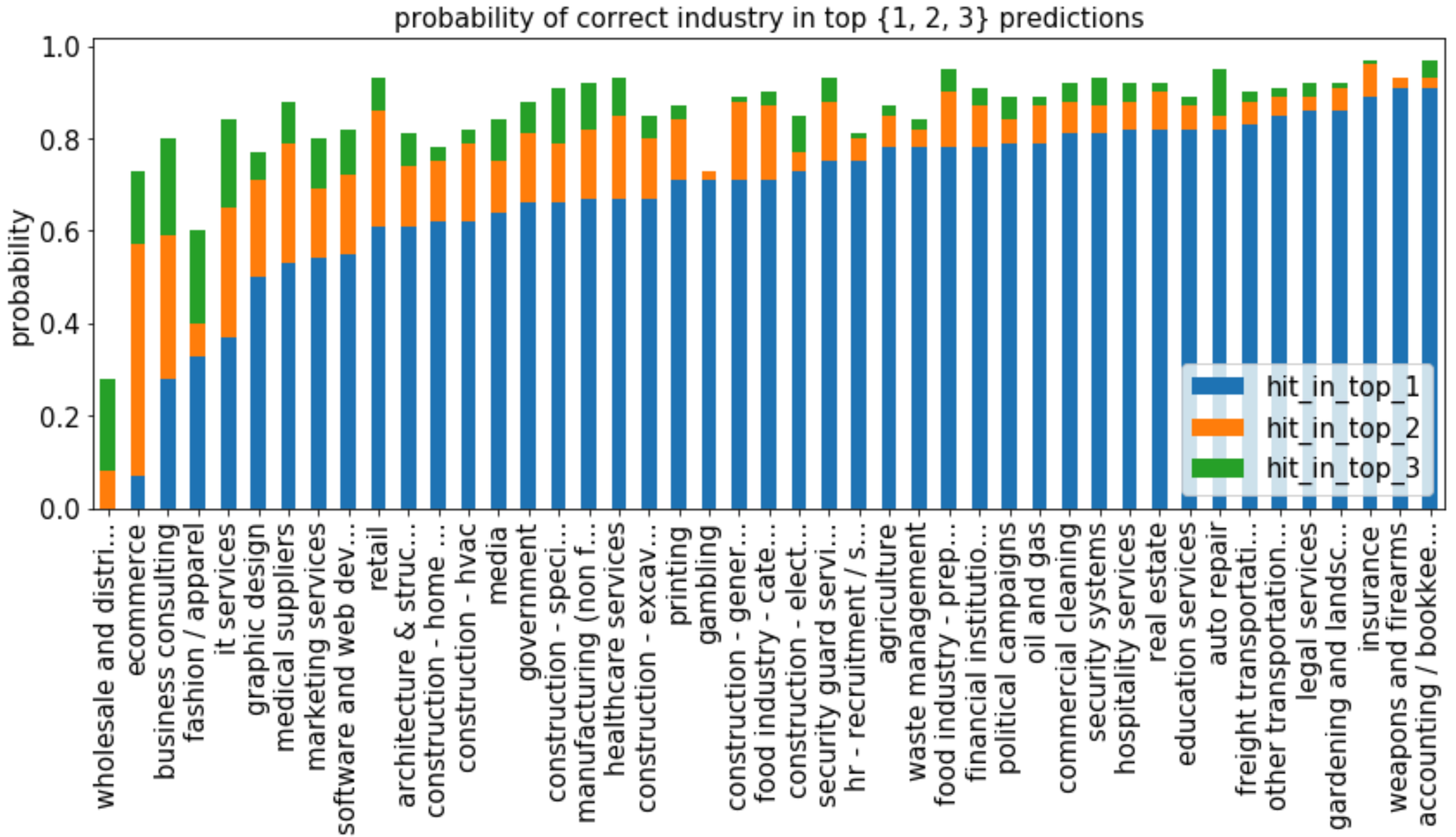
Web Model - Design

TF-IDF MODELS - MLP NN - COSINE SIMILARITY

```
def l2_activation(x):
    return K.l2_normalize(x, axis=1)

get_custom_objects().update({'custom_activation': Activation(l2_activation, name='l2_activation')})

input_layer = Input(shape=(data_input_dim,), sparse=True)
x = Dense(first_layer_input_dim, activation='relu')(input_layer)
x = Dense(3000, activation='relu')(x)
x = Dense(300, activation='relu')(x)
x = Dense(128, activation='relu')(x)
x = Activation(l2_activation)(x)
x = Dense(num_pred_categories, activation=l2_activation)(x)  # DOT PRODUCT + L2 NORMAZLIATION = COSINE SIMILARITY
out = Dense(num_pred_categories, activation='softmax')(x)

model = Model(inputs=input_layer, outputs=out)
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])
```

Web Model - Performance

PRELIMINARY RESULTS



Web Model - Performance

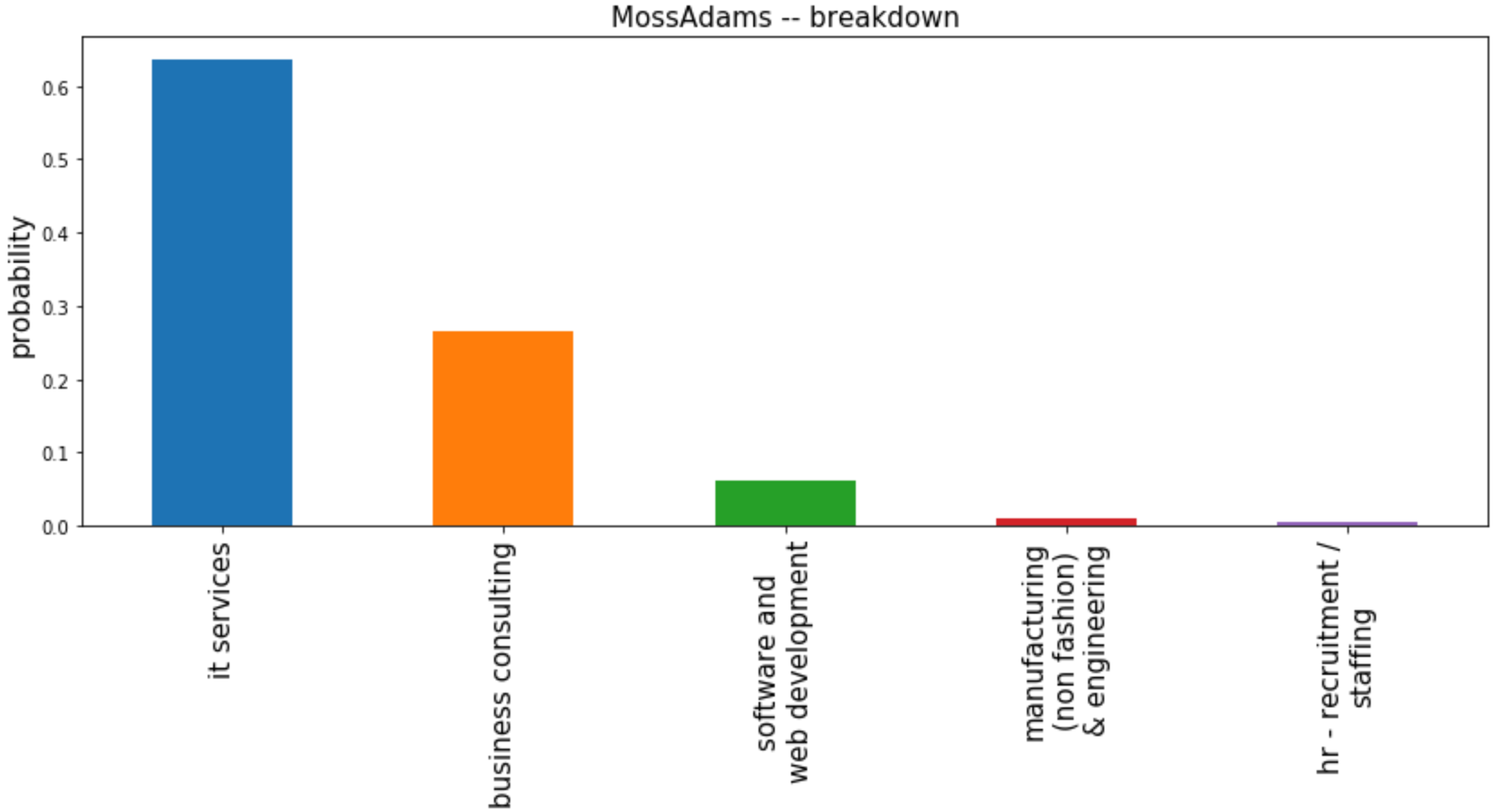
CHALLENGE - MULTIPLE “RIGHT” ANSWERS

Example: https://www.mossadams.com/ — “IT Services” or “Business Consulting”?

Web Model - Performance

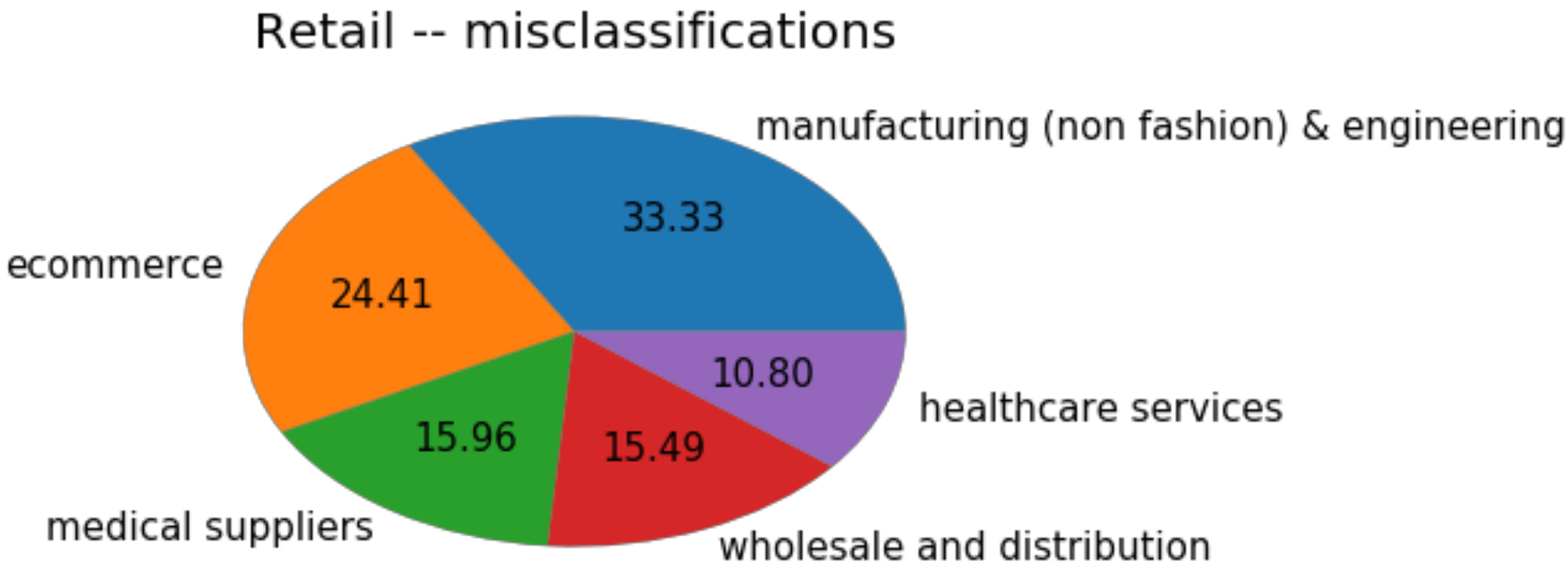
CHALLENGE - MULTIPLE “RIGHT” ANSWERS

Example: https://www.mossadams.com/ — “IT Services” or “Business Consulting”?



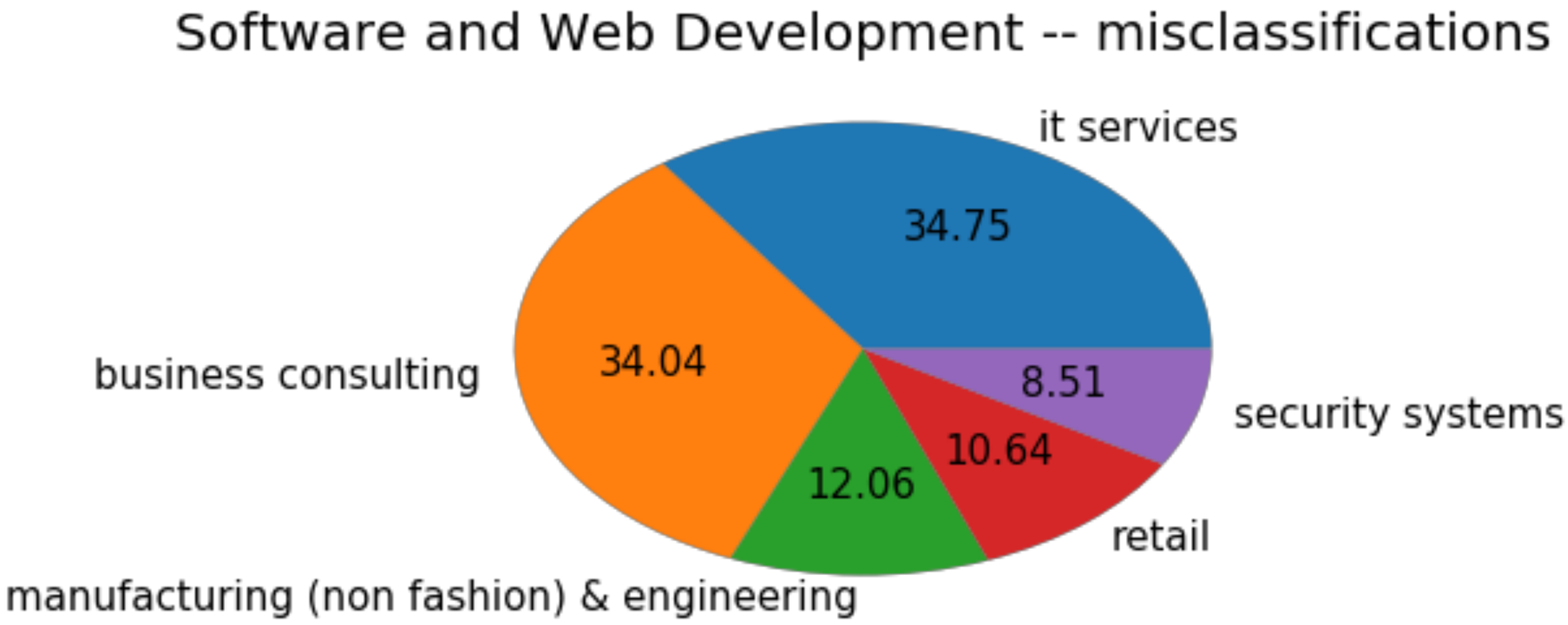
Web Model - Performance

CHALLENGE - AMBIGUOUS DEFINITIONS



Web Model - Performance

CHALLENGE - AMBIGUOUS DEFINITIONS



Web Model - Performance

UPCOMING IMPROVEMENTS

- Use 100% of the dataset.

- Optimize & QA.

- Integrate additional features in the layer-2 model
  - URL text features
  - Metadata features (page count, word count, etc)

- Integrate with additional sources
  - Other 4 external sources.
  - Invoice data

Web Model - Use Cases

EXAMPLE - SEGMENTATION

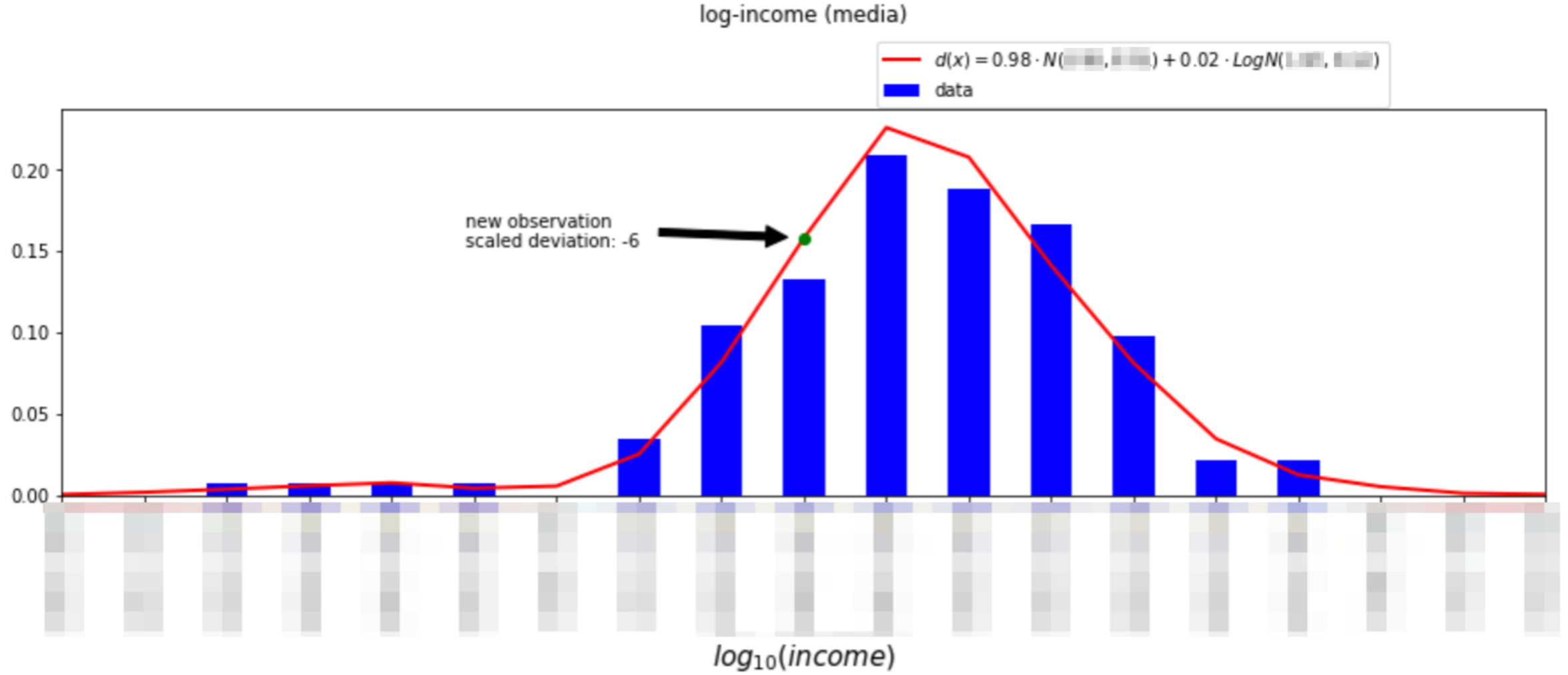
- Select a metric (e.g. income, credit score, web presence).

- Model the distribution of that metric
  - Across all clients
  - For every specific industry

- Compare and contrast - is it normal? too high? too low?
  - A client against all clients
  - A client against all clients within his industry

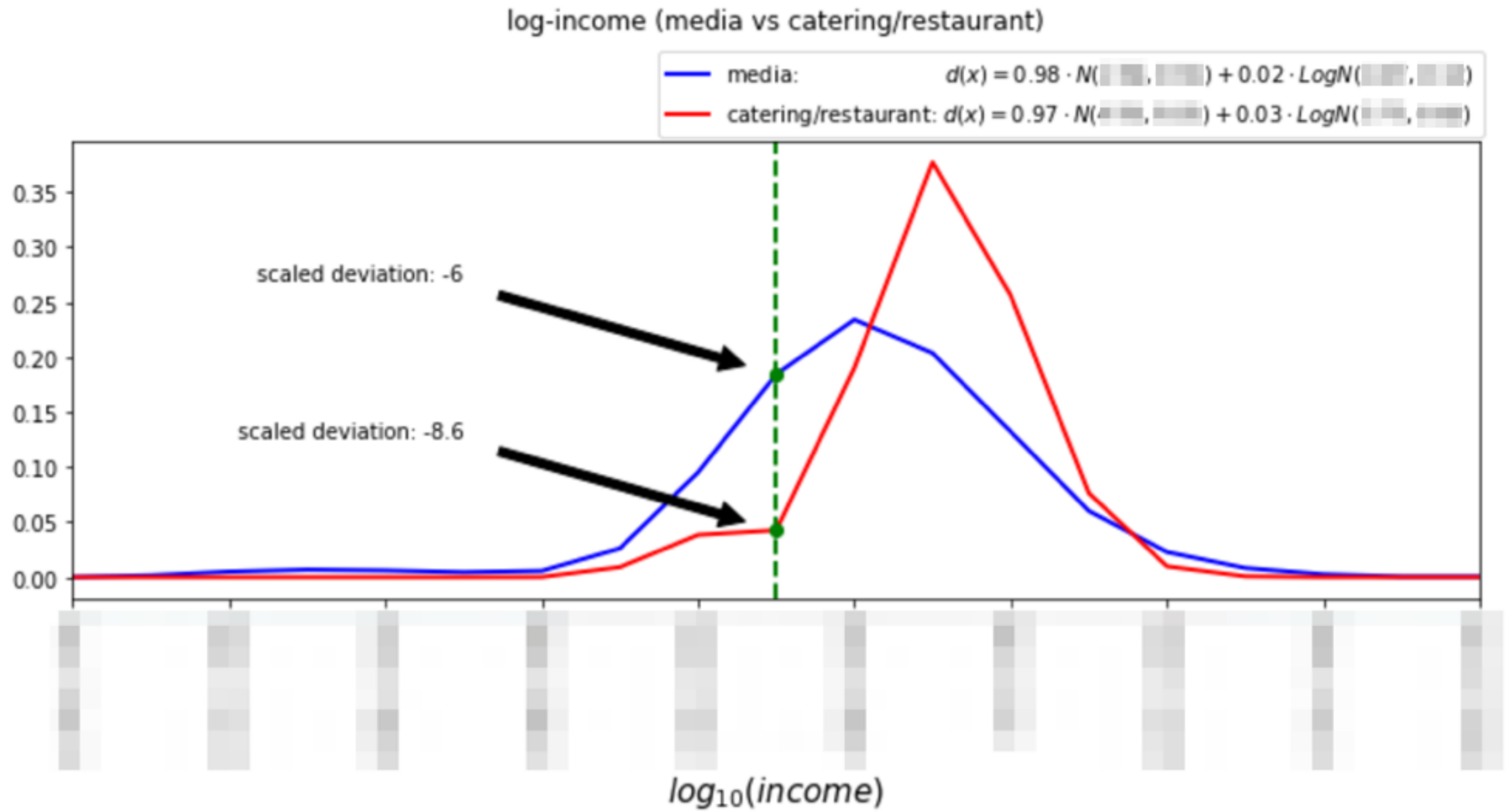
Web Model - Use Cases

EXAMPLE - SEGMENTATION



Web Model - Use Cases

EXAMPLE - SEGMENTATION



Key Take Aways

- When starting a classification project from zero, think very hard about the classes and their definitions. It's **MUCH** harder to correct in later stages.

- When working with crowdsourcing platforms, put a lot of effort into QA and monitoring. **Garbage in — Garbage out!**

- Optimize your training code for efficiency. Waiting hours / days for a process to finish can seriously disrupt the development process.
  - + some may be useful for the production code as well.
  - + learn some cool stuff along the way.

Key Take Aways

- Use off-the-shelf libraries where possible. **Don’t reinvent the wheel!**

- When using libraries, look first for the “high level” ones that are easy to use (e.g try Keras before going to TensorFlow).

- For heavy ML work, forget your laptop… **use cloud services**!
  - For text processing — computing / memory optimized instances (e.g. Amazon C3 / C4 / R4).
  - For deep learning — GPU instances (e.g. Amazon P2).

- For you PyCharm users — check out remote development feature. It work’s amazingly well.

References & Useful Links

- **BlueVine Data Science Blog (coming soon!): https://medium.com/bluevine-data-science**
- **BlueVine public Git repo (coming soon!): https://github.com/bluevine**

- Convolutional NN's: link1 (ODSC blog), link2 (Stanford course), link3 (nice slideshow)
- TF-IDF: link1 (StackOverflow)
- Scrapy: link1 (Scrapy bugs Git), link2 (middleware)
- Spacy: link1 (Spacy in parllel)
- Stacking models: link1 (KDNuggets), link2 (Kaggle Blog)
- GPU monitoring: link1 (StackExchange)
- NLP: link1 (hashing trick), link2 (word2vec tutorial), link2 (Analytics Vidhya), link3 (lemmatization)
- Keras: link1 (StackOverflow), link2 (rant on TensorFlow)
- Remote development: link1 (using Jupyter), link2 (using Pycharm)

We’re Hiring!

Contact me here: ido.shlomo@bluevine.com

Check out the “Data Scientist” listing on Lever: https://jobs.lever.co/bluevine