Analysis and Prediction of Application Categories on Online Application Stores

Abstract

We explore quality control problem related to online application stores: automating reviews of submitted applications and organizing online store using categories. We approach both problems by predicting application categories using application descriptions and describe potential positive and negative outcomes of our approach.

We discuss suitable multi-class and binary classification methods and describe three experiments we conduct, including details about data preprocessing, validation, testing and the final results. For evaluating our approach, we use an F-measure and Confusion Matrix. As the result, we were able to identify confusing categories that should be taken into consideration.

1. Introduction

Mobile applications play an important role in the modern information society. The number of official applications exceeds several hundred thousands (Reisinger, 2012), and this number keeps increasing. Official applications are published on online stores like Google Play Store[1], Apple App Store[2], Windows Phone Store[3], or Nokia Ovi Store[4]. In Google Play store, for example, there are more than 800,000 applications as of February 2013, and this number dramatically increased since 2009 when the store has been created (Wauters, 2009).

Application stores act like online shops, and involves several stakeholders. Developers are "sellers", they create mobile applications and publish them. Online store managers organize the list of applications to make them easy to find and access. Mobile phone users, "consumers", download or buy applications either visiting the store's website, or using their mobile device capabilities.

In the conditions of market growing and several stakeholders' interest there is a question of quality. Users want a software of good quality, which meets the specifications and follows official rules. Developers want their work to get easily published, be accessible and profitable. Online store managers want to promote the platform (Android, IPhone, etc.) and motivate developers and users to publish and download applications, respectively.

1.1. Problem Description

One big issue we are addressing is quality control of applications published on an online store. Online application stores have strict regulations and instructions the developers need to follow. However, there are cases of malicious applications (Matteson, 2013), and applications of low quality (Victor, 2013). The problem of a low quality relates not only to the application code, but to product details too: the description must be clear and concise, and the application should be easily found on the store by its category. Some stores like Apple App Store and Windows Marketplace have a mandatory official review procedure, which leads to a better quality. However, this procedure is mostly done manually, which is costly and leads to a time-to-market delay. On the other hand, applications on Google Play store can be published right away, which saves time and review efforts, but leads to the overall worse quality. Google has to remove faulty applications from time to time to make the store clean (Victor, 2013). A better option in both cases would be semi-automated reviews.

Another big issue is the overall organization of an application store. All online stores allow to observe new or most popular applications, as well as to find an application by a given name or by looking through categories. However, there are problems that have not

[1] http://play.google.com/store
[2] http://www.apple.com/iphone/from-the-app-store/
[3] http://www.windowsphone.com/store
[4] http://store.ovi.com/

Project report for CS886: Applied Machine Learning. University of Waterloo, Spring 2013.

been addressed well for these types of stores. First, it is common that a mobile application is developed for different platforms, however, different online stores have different category hierarchies. For example, Windows App Store has a category `Government and Politics`, while Google App Store does not. Next, some categories do not have exact boundaries, and therefore, may be misleading. For instance, the `Entertainment` category can overlap with `Games`, `Media and Video`, `Music and Audio`, but stores do not allow assigning multiple categories. And finally, there is another opportunity of fraud: an application can be intentionally put into a more visited, but wrong category, or into multiple categories with a different titles and package names to increase developer's profits.

1.2. Our Approach and Expected Outcomes

We propose an approach which can push the research forward in addressing the two big issues described above. We are going to explore a relationship between an application's category and other parameters like title and textual description in the context of the online application store - Google Play store. In case we are able to make a good category prediction based on textual descriptions and title of application, we see positive expected outcomes as follows.

1. Categories on an online stores are clear and easy to distinguish. This means that online stores are organized properly, and the idea of a single category works well in this case.
2. Automated category-checking is possible. Moreover, if a category is a completely derived property, there is no need to specify it manually.

In case our methods do not work, the following negative outcomes are highly probable.

1. Categories on an online stores do not have exact boundaries. This leads to confusion from users' point of view (where to search an application) and developers' one (where to put an application).
2. Categories are not derived properties and have to be specified by developers and properly reviewed manually by online store managers.

1.3. Defining a Machine Learning Problem

In this work, we are trying to predict a category, which can be one of several (32) category names. This is a multi-class classification problem, and the target is the variable **category** which can take of 32 values. The input variables are features that have to be derived from textual descriptions.

2. Related Work

There is a related work (Zhu et al., 2012) that explores classification of mobile applications using Web search based on its title, but for a different purpose. The authors analyze logs of mobile applications users and explore which type of applications are used: games, social networks, etc. Their approach assumes predefined common-sense taxonomy that is not the same as the actual categories of application stores. Their prediction has been evaluated on a set of 8,852,187 usage records, but 680 different applications, which cannot be claimed to be working for all applications. In our case, we have all available data in our dataset with greater number of applications, so that we can do data mining, prediction and evaluation over this dataset.

Another work addressed mobile application classification with the usage of static analysis (Shabtai et al., 2010). The authors were able to distinguish games and tools with a very high prediction. This is a good support for our hypothesis, because it may imply that categories are derived properties. However, they evaluated their work on 2850 applications only, which is hard to generalize, in our opinion.

3. Description of Available Data

The private data source (Dienst & Berger, 2012), contains 625,067 versions of 281,079 unique Android applications published on Google Play Store within 2011. Each version is considered as an instance Tab. 1. We consider using **full textual description** to derive input variables and **category** as an output variable.

app title	Ivy Leaf Live Wallpaper
type	APPLICATION
full textual description	A beautiful Ivy Leaf live wallpaper which lets you interact with falling leaves. If you like this wallpaper, please support us by buying Pro version with more themes and features...
category	`Personalization`
version	1.0.2
...	...

Table 1. An instance in the dataset

4. Data Preprocessing

We cannot use the dataset described above cannot directly for our work. Therefore, we perform filtering to remove all unwanted instances and feature construc-

tion to define input features for machine learning.

4.1. Filtering

First, we extract only the latest versions of each application. One reason is that multiple versions can have similar descriptions and introduce bias. The second reason is that the latest versions are more probable to have a longer description, since they often include the list of recent changes, which might tell something about the category.

Next, we focus only on English versions of applications. Some applications have multiple versions in several languages, and in this case we took English versions only. However, the applications without English versions are omitted. The reason of this filtering is to focus our methods more and do not take into account language-specific features.

Next, we consider only instances of the the type `Application`, not `Games`, since we assume that categories matter less for games and it is harder to distinguish various game genres by description.

An finally, we notice old categories that do not exist anymore. For instance, the categories `Software Libraries` and `Demos` existed in past and have been merged into a single one - `Libraries and Demos`. However, we do not have a clear evidence of this merge and the way it is done, but our approach may identify the confusion between several categories. So, at this step, we do not filter out any categories.

The final filtered dataset statistics is shown on Tab. 2.

#	Category	Instances	From Total
1	books_and_reference	7993	4.30%
2	business	8005	4.31%
3	comics	4814	2.59%
4	communication	5566	3.00%
5	demo	939	0.51%
6	education	7263	3.91%
7	entertainment	23531	12.66%
8	finance	4293	2.31%
9	health	1087	0.59%
10	health_and_fitness	4092	2.20%
11	libraries_and_demo	3626	1.95%
12	lifestyle	10154	5.47%
13	media_and_video	5920	3.19%
14	medical	2077	1.12%
15	multimedia	1345	0.72%
16	music_and_audio	16163	8.70%
17	news_and_magazines	6061	3.26%
18	news_and_weather	1436	0.77%
19	personalization	11800	6.35%
20	photography	6066	3.26%
21	productivity	6169	3.32%
22	reference	1600	0.86%
23	shopping	3739	2.01%
24	social	6644	3.58%
25	software_libraries	368	0.20%
26	sports	8742	4.71%
27	themes	1557	0.84%
28	tools	13961	7.51%
29	transportation	1885	1.01%
30	travel	1452	0.78%
31	travel_and_local	6408	3.45%
32	weather	1040	0.56%
	TOTAL	185796	100.00%

Table 2. The dataset statistics after the filtering: categories and the number of instances in each category.

4.2. Feature Construction

An important step is to derive features from textual descriptions. We use a common "bag of words" method tailored to our multi-class classification problem. To derive features, we consider only a training set, because we cannot assume we have any information about validation and training sets.

The performance of classification algorithms depends on the number of features, therefore, we found useful to infere a certain number N of the top features. Also we denote the number of categories as C, and $C = 32$ in our case. Since we have several categories and instances within each of categories should have fair opportunities, the number of text features we derive is proportional on the number of instances in each category.

Each of N features is a boolean (`True`, `False`) meaning whether a certain word occurs in a description. Formally, for the instance X_i and its textual description T_i, the feature value $F_{i,j} = IsPresent(A_j, T_i)$, where A_j is a word, $1 \leq j \leq N$, and $IsPresent$ is a boolean function (`True`, `False`) meaning whether the word occurs in a given description T_i.

We derive our features as follows.

1. We preprocess text descriptions by making all words lowercase, removing all stop words, stemming and replacing commonly-used patterns. We use a list of stop words offered by NLTK (see Sec. 6.2) augmented by the list of MySQL list of stop words (Ranks.NL). We use the NLTK's stemming class `nltk.stem.snowball.EnglishStemmer` to extract roots from all words (e.g., "traveling" to "travel"). Also, we completely remove the collocation "Recent changes" and replace collocations representing age restriction ("Content rating: low maturity") by a single word ("low_maturity").

2. For each category, we concatenate all the normalized text descriptions of instances within this category, into a big list of words. Surely, this list is not necessarily a set, and may contain repetitions. So, we have got C lists of words.

3. In each list, we take N most frequently occurring words. We can take less, since N is the final number of features, but it is easy to get just N.

4. We define an empty set S to represent the final set of words that define features.

5. We loop through all the categories, derive features from textual descriptions of instances of each category proportionally to the number of instances in the category, and then merge it with S.

6. If we arrive at the number of features less than N (since some of words repeat), we go to the last step again until we get as close as possible to N.

During our validation (K-fold cross validation) we have a different training set each fold, therefore, we have to derive features every time again. This increases execution time, however, we can validate our feature selection during our validation, which is a good possibility.

Once we have features from our training set, we convert all the descriptions from the training set, validation set and test set into feature values.

5. Classification and Analysis Methods

In this section we discuss multi-class classification and binary classification methods that are applicable in our approach. Also, we describe evaluation techniques we use to measure the performance of our prediction.

5.1. Multi-Class Classification

Multi-class classification is often considered as generalization of binary classification. We apply the following most common approaches (Rifkin, 2008).

- **One-versus-all classification**. For C categories, C binary classifiers are trained in total, one classifier per category.
- **One-versus-one**, or **round-robin classification**. For C categories, $C*(C-1)/2$ binary classifiers are trained for each pair of C categories.

The first approach is simpler, and we ran preliminary experiments using One-versus-all classification. However, we get the following problems:

1. The dataset is skewed, since some categories have small number of instances (like `demo` and `software_libraries`, Tab. 2), compared to the "rest" instances. This makes the evaluation hard, since the accuracy is almost 99% because it is more likely to classify an instance to the "rest" of the set and guess correctly. More precise measures like precision, recall and F-score were 20%, 30%, and 24%, respectively, which is very low.

2. If some categories are confused with other ones (like `demo` and `libraries_and_demo`), we have only confusion matrix (see evaluation), but we may need binary evaluations too.

Because of these two reasons, **we use the round-robin classification method** in all our experiments.

One common way of constructing a multi-class classifier is combining binary ones using a voting mechanism. We use it as follows:

1. Each binary classifier tries to classify an instance of any class (even the one the classifier does not cover) and assigns probabilities to each class, e.g., `communication`: 60%, `travel`: 40%. Probabilities are useful, because they give less certainty to a particular class if the classifier actually does not cover a particular class.

2. The final class is the one with the highest score—the sum of all produced probabilities for the given class. For example, if we have three classifiers that gave 14%, 98% and 80% to `communication`, the score of the `communication` category is 192%. The final decision is made after comparison and calculation of all categories' scores.

5.2. Binary Classification Methods

We propose the usage of the following methods for our problem: *Naïve Bayes* (S. Russel, 2002), *Maximum Entropy*(Berger et al., 1996) and *Support Vector Machines*(N. Cristianini, 2000).

Naïve Bayes method has an advantage of fast training. Moreover, this method has been successfully used for spam filtering (Sahami et al., 1998), which is a text classification problem also. The required feature independence is not always true: the word "send" may occur jointly with the word "message", and the classification method will consider them separately, causing double-counting. However, we ignore these cases and assume the problem of double counting will not decrease the performance.

Maximum Entropy, in contrast, takes into account cases of feature interactions, which is useful in text classification. However, this method is iterative and very slow.

We do not use *Support Vector Machines*, because they require some tuning, which is not feasible within the time constraints. This remains as a future work.

5.3. Evaluation

For validation and testing of multi-class classification, we use an *F-Measure* and a *Confusion Matrix*.

A *Confusion Matrix* (Kohavi & Provost, 1998) is a useful evaluation of multi-class classification problems. Each cell $C_{i,j}$ of a confusion matrix shows the number of instances when the predicted label is j while the correct label is i. This is particularly useful in our domain, because we want to identify similar categories, which are, therefore, confused when predicting.

A *Micro-Averaged F-score*, or *F-measure* (Asch, 2013) is a performance measure for multi-class classification problems (including multi-label classification). In our case, the F-score can be computed from a confusion matrix as follows. The sum of all true positives TP is the sum of all diagonal elements of the matrix. Since we have a single-label multi-class classification, the number of all false negatives FN equals the number of all false positives FP and equals the sum of all non-diagonal elements of the confusion matrix. The micro-averaged F-score equals overall precision and recall and equals: $F_\mu = \frac{TP}{TP+FP} = \frac{TP}{TP+FN}$

For binary classifiers, we record *precision*, *recall*, an *F-score* and the most informative features during validation. However, we do not use any of these for validation purposes: we may need them for a more detailed analysis if our approach fails.

6. Experiment Design

In this section we show an experiment template and tools used for implementing our experiments.

6.1. Experiment Template

All our experiments follow the same template described below. The variability among the experiments is defined by the following parameters: K - the number of partitions used in K-fold cross validation; CTS - a set of categories among which we classify instances; and BC - a binary classifier used in the experiment.

We run a K-fold cross-validation with 40%-training set, 40%-testing set and 20%-validation set. We derive $N = 1000$ features by considering all classes from CTS. The purpose of our validation is **deriving the best features** (rather than parameters of a classifier) that make a given multi-class classifier be the most effective on our dataset preprocessed using the features. Each iteration (each fold), we train several binary classifiers of the type BC and evaluate them separately. Then, for the current fold, we combine them all into a multi-class classifier and evaluate and test it. At the end, we use the best validation measure (in our case, it is an F-measure) to choose the best classifier. The corresponding testing result is the actual testing evaluation of the final multi-class classifier, and the features used for this classifier are also included in the final result. Below we present the flow in a more formal way.

1. For each fold number f out of K:
   (a) Partition the dataset into the training, validation and testing sets.
   (b) Construct features from the whole training set as described in Sec. 4.2.
   (c) For each two categories t_i; $t_j \in CTS$:
      i. Train a binary classifier BC_f, ij on the subset of the training set that contains only instances of categories t_i and t_j.
      ii. Test the constructed binary classifier on the validation set's subset that contains only instances of categories t_i and t_j.
      iii. Record the binary classifier BC_f, ij.
   (d) Build a round-robin multi-class classifier MC_f (as described in section Sec. 5.1) from the binary classifiers BC_f, ij.
   (e) Validate MC_f on the whole validation set and record all the multi-class evaluation data (Sec. 5.3) denoting it as $MCVal_f$.
   (f) Test MC_f on the whole test set and record all the multi-class classification evaluation data (Sec. 5.3) denoting it as $MCTst_f$.
2. Among all classifiers MC_fs, choose the multi-class classifier MC_b with the best $MCVal_b$.
3. The final classifier is MC_b, and the final testing result is $MCTst_b$.

6.2. Tools and Libraries

We use Python version of the *Natural Language Processing Toolkit (NLTK) 2.0*[5]. We implement feature construction using `nltk.probability.FreqDist` for getting word occurrence frequency and implement multi-class classification using the included binary classifiers `nltk.classify.maxent.MaxentClassifier` and `nltk.classify.naivebayes.NaiveBayesClassifier`. We use `nltk.metrics` for evaluation, including `nltk.metrics.confusionmatrix.ConfusionMatrix`.

[5] http://nltk.org/

7. Experiments and Results

Using the experiment template defined in Sec. 6.1, we run three experiments: small experiments A and B in order to evaluate different binary classifiers on the same subset of our dataset; and a big experiment C that is aimed to evaluate the best classifier on the entire dataset. We took the F-measure=60% as the threshold of all our experiments.

7.1. Experiment A: Naïve Bayes, 4 categories

The first experiment we run is as follows: $K = 5$, $CTS = \{$`music_and_audio`, `tools`, `personalization`, `lifestyle`$\}$ and $BC =$ `nltk.classify.naivebayes.NaiveBayesClassifier`. The resulting F-measure is 0.766202592415, or 76.6%. The resulting (printed after testing) confusion matrix is shown on Tab. 3. The performance is above the threshold, which means the success: the classifier can distinguish between these four categories.

#	categories	01 music_and_audio	02 tools	03 personalization	04 lifestyle
01	music_and_audio	**4911**	553	681	320
02	tools	139	**4759**	311	375
03	personalization	123	385	**4067**	145
04	lifestyle	258	989	591	**2223**

Table 3. Confusion Matrix for Experiment A. Row headers denote actual categories, column headers denote predicted categories. The bold are diagonal elements - the number of instances guessed correctly (true positives)

7.2. Experiment B: MaxEnt, 4 categories

This experiment is defined as follows: $K = 5$, $CTS = \{$`music_and_audio`, `tools`, `personalization`, `lifestyle`$\}$ and $BC =$ `nltk.classify.naivebayes.MaxEntClassifier`. The resulting F-measure is 0.824060294753, or 82.4%. The resulting confusion matrix clarifies the result (Tab. 4). The experiment is successful, as the performance is even better by around 6% compared to the experiment A. The classifier can distinguish between these four categories as well. However, the current experiment took around 20 hours to run on our machine, therefore we concluded that it is not feasible to use this classifier on any bigger set of observations. For the following experiment we use the Naïve Bayes classifier as it is fast and still has good performance.

#	categories	01 music_and_audio	02 tools	03 personalization	04 lifestyle
01	music_and_audio	**5884**	224	152	205
02	tools	119	**4722**	183	560
03	personalization	266	341	**3947**	166
04	lifestyle	199	904	346	**2613**

Table 4. Confusion Matrix for Experiment B. Row headers denote actual categories, column headers denote predicted categories. The bold are diagonal elements - the number of instances guessed correctly (true positives)

7.3. Experiment C: Naïve Bayes, 32 categories

We tried this big experiment with $K = 10$, $CTS = \{$`all categories`$\}$ and $BC =$ `nltk.classify.naivebayes.MaxEntClassifier`. The experiment was taking a long time to run, therefore, we have terminated it after the first fold. The resulting F-measure is 0.402510392703, or 40.3%. The resulting confusion matrix is shown on Tab. 5.

The performance here is below the threshold, which implies the negative outcomes of our work. By looking at the confusion matrix (Tab. 5) we see that the overall performance has been significantly decreased if certain categories were confused a lot.

First, the reason is in categories that are "too generic". For example, the category `multimedia` has been correctly guessed only 12 times, while it has been confused mostly with the `entertainment` category: 120 times. If we look at `entertainment`, it has been mostly guessed correctly, however, it has been confused with other categories, including `photography`. The instances of category `lifestyle` were very often misclassified. This brings us to the first conclusion: some categories are general and are often confused with more specific ones, therefore, this implies user and developer confusion when they choose between these kind of categories. In contrast, the categories `media_and_video` and `music_and_audio` were not confused very often as it might seem, this implies that they are quite clear to distinguish.

Next, instances in our training, test and validation sets are not uniformly distributed by categories, and some categories have very small number of instances. For instance, the dataset contains just several hundreds instances that belong to `software_libraries`. Therefore, it was harder to train and evaluate all the classifiers that consider this category. By looking at the binary classification results, we see that the F-measure

#	all categories	01 entertainment	02 productivity	03 music&audio	04 personalization	05 sports	06 tools	07 photography	08 travel&local	09 books&refer.	10 education	11 media&video	12 news&magaz.	13 business	14 communication	15 lifestyle	16 social
01	entertainment	**2385**	788	513	811	487	181	1144	404	360	378	976	164	46	173	73	219
02	productivity	34	**1516**	19	63	32	179	18	58	58	62	29	20	51	123	3	16
03	music&audio	327	295	**3850**	61	132	55	283	112	92	78	646	44	9	62	3	36
04	personalization	172	405	34	**2514**	85	63	862	17	71	12	97	3	.	31	15	12
05	sports	51	152	16	40	**2645**	28	63	106	25	53	30	85	4	10	4	28
06	tools	160	2652	65	130	77	**1061**	51	178	145	124	112	16	37	312	7	26
07	photography	79	223	21	170	50	15	**1431**	37	60	11	66	19	6	9	6	47
08	travel&local	35	152	21	27	39	31	27	**1572**	80	41	12	102	18	18	2	18
09	books&reference	135	308	69	43	65	106	116	143	**1462**	175	23	26	67	43	18	48
10	education	101	281	30	31	112	101	46	183	271	**1459**	29	84	32	23	2	22
11	media&video	175	335	93	56	48	45	226	32	35	32	**1081**	31	9	28	1	23
12	news&magazines	42	182	33	10	328	19	56	65	63	22	34	**1316**	26	24	6	24
13	business	105	589	86	24	136	85	10	372	110	72	25	138	**857**	74	14	47
14	communication	105	608	21	48	41	88	16	54	50	32	22	31	39	**878**	3	108
15	lifestyle	255	599	84	195	134	103	212	570	314	262	84	120	90	94	**227**	169
16	social	134	398	53	93	162	32	65	173	92	74	49	70	31	251	17	**889**
17	finance	37	246	7	8	22	18	3	29	29	19	13	32	39	19	2	4
18	health&fitness	71	196	17	30	45	34	52	70	51	45	28	14	15	14	28	19
19	comics	141	107	14	246	14	7	420	9	151	23	52	4	3	27	5	21
20	shopping	77	238	35	19	57	44	15	238	27	13	10	14	43	16	12	15
21	libraries&demo	66	214	14	89	34	108	331	19	42	38	39	2	9	34	5	5
22	transportation	17	108	15	10	5	17	4	147	4	8	1	4	30	16	.	5
23	medical	31	74	4	5	17	9	11	32	45	38	8	4	13	10	1	4
24	weather	46	26	15	16	8	7	3	24	4	2	2	11	3	3	.	.
25	themes	25	61	5	256	18	7	39	2	.	.	13	.	.	5	.	6
26	news&weather	12	34	2	1	63	8	2	20	10	1	2	178	5	5	.	7
27	reference	48	86	3	.	31	17	11	38	156	63	23	11	6	22	13	25
28	travel	14	68	8	4	7	17	16	229	15	9	2	2	.	16	3	2
29	multimedia	120	80	53	24	16	24	80	4	14	7	62	4	1	17	1	9
30	health	43	80	1	4	12	21	5	15	10	11	3	5	5	5	11	6
31	demo	52	46	5	8	17	59	3	8	36	15	5	3	2	12	2	3
32	software_libraries	6	28	4	5	1	10	16	2	7	3	2	.	4	4	.	2

#	all categories	17 finance	18 health&fitness	19 comics	20 shopping	21 libraries&demo	22 transportation	23 medical	24 weather	25 themes	26 news&weather	27 reference	28 travel	29 multimedia	30 health	31 demo	32 software_librar.
01	entertainment	21	101	35	42	26	34	19	21	7	1	1	.	1	.	.	2
02	productivity	65	27	4	19	8	40	3	20	.	.	.	.	.	1	.	.
03	music&audio	2	62	28	6	178	6	1	5	93	.	.	.	.	.	.	.
04	personalization	4	23	104	1	112	1	1	41	40	.	.	.	.	.	.	.
05	sports	8	64	.	5	.	48	5	18	5	.	1	1	1	.	.	1
06	tools	113	72	2	22	11	102	6	97	.	.	2	1	.	2	.	2
07	photography	1	11	138	3	1	5	6	.	12	.	.	.	.	.	.	.
08	travel&local	16	10	.	12	1	292	3	34	1	.	.	.	.	.	.	.
09	books&reference	22	64	43	12	155	25	20	9	.	.	1	.	.	.	.	.
10	education	20	28	1	6	4	6	19	14	1	.	.	.	.	.	.	.
11	media&video	2	7	2	.	89	4	4	7	2	.	.	.	1	.	.	.
12	news&magazines	16	6	3	5	15	11	33	37	20	29	.	.	.	.	.	.
13	business	226	33	.	99	2	34	58	6	.	.	.	.	.	.	.	.
14	communication	38	12	.	6	1	13	3	5	3	.	1	.	.	1	.	.
15	lifestyle	80	197	7	134	16	44	25	38	.	.	3	.	2	3	.	1
16	social	18	14	2	19	1	10	7	3	.	1	.	.	.	.	.	.
17	finance	**1152**	8	.	15	.	5	3	8	.	.	.	.	.	.	.	.
18	health&fitness	27	**757**	2	10	.	16	60	11	25	.	.	.	.	.	.	.
19	comics	2	11	**569**	2	92	1	1	.	4	.	.	.	.	.	.	.
20	shopping	50	11	.	**536**	2	9	1	4	10	.	.	.	.	.	.	.
21	libraries&demo	6	8	61	3	**294**	8	11	7	.	.	.	.	.	.	.	4
22	transportation	7	4	.	8	.	**326**	5	13	.	.	.	.	.	.	.	.
23	medical	11	123	.	6	1	2	**377**	5	.	.	.	.	.	.	.	.
24	weather	5	1	.	.	.	6	2	**231**	1	.	.	.	.	.	.	.
25	themes	2	.	3	1	.	.	.	3	**175**	1	.	1	.	.	.	.
26	news&weather	5	1	2	.	.	2	1	43	.	**170**	.	1	.	.	.	.
27	reference	7	14	1	3	1	19	2	8	.	.	**31**	.	.	1	.	.
28	travel	9	1	.	.	.	144	1	9	.	.	1	**4**	.	.	.	.
29	multimedia	3	.	3	.	.	.	.	3	1	.	.	.	**12**	.	.	.
30	health	5	142	.	4	.	5	21	3	.	.	.	.	.	**18**	.	.
31	demo	1	4	.	.	3	4	3	.	.	.	1	.	1	1	**82**	.
32	software_libraries	9	.	1	.	.	.	1	1	.	.	.	.	.	.	.	**42**

Table 5: Experiment C: Confusion Matrix. Row headers - actual categories, column headers - predicted categories.

for `software_libraries` is 42%, which is low, while for `music_and_audio` it is 66%. So, we could get better results if our dataset contains uniformly big number of instances within each category.

8. Conclusions and Future Work

From the Machine learning viewpoint, we have tested round-robin classification with two binary classifiers: Maximum entropy and Naïve Bayes on classification of mobile applications' descriptions. The former method gave s slightly better performance, but the performance of both binary classifiers on a subset of four categories was satisfactory. We have conducted a big experiment with all categories, which turned to be below the threshold. The possible reasons are confusing categories, as well as the skewness of our dataset.

From the practical viewpoint, we have identified risky categories: `multimedia`, `entertainment` and `lifestyle` that should be considered very carefully by online-store managers, as well as developers and users. We were unable to prove or disprove that categories are derived properties of mobile applications or should be specified.

The future work includes exploring other classifiers like SVM in this domain as well as conducting the experiment C using Maximum Entropy Classifier.

References

Asch, Vincent Van. Macro- and micro-averaged evaluation measures, June. 2013. URL http://www.cnts.ua.ac.be/~vincent/pdf/microaverage.pdf.

Berger, Adam L., Pietra, Vincent J. Della, and Pietra, Stephen A. Della. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22 (1):39–71, March 1996. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=234285.234289.

Dienst, Steffen and Berger, Thorsten. Static analysis of app dependencies in android bytecode, 2012. Tech. Note, available at http://www.informatik.uni-leipzig.de/~berger/tr/2012-dienst.pdf.

Kohavi, R. and Provost, F. *Glossary of terms, Machine Learning*. 1998.

Matteson, Scott. Malware in the google play store: Enemy inside the gates. May 2013. URL http://www.techrepublic.com/blog/google-in-the-enterprise/malware-in-the-google-play-store-enemy-inside-the-gates/.

N. Cristianini, J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000. ISBN 0-521-78019-5.

Ranks.NL, Webmaster Tools. English stopwords. URL http://www.ranks.nl/resources/stopwords.html.

Reisinger, Don. Can Apple's App Store maintain its lead over Google Play?, Sep. 2012. URL http://news.cnet.com/8301-1035_3-57521252-94/can-apples-app-store-maintain-its-lead-over-google-play/.

Rifkin, Ryan. Multiclass classification, Feb. 2008. URL http://www.mit.edu/~9.520/spring09/Classes/multiclass.pdf.

S. Russel, P. Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, 2002.

Sahami, Mehran, Dumais, Susan, Heckerman, David, and Horvitz, Eric. A Bayesian Approach to Filtering Junk E-Mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.1254.

Shabtai, A., Fledel, Y., and Elovici, Y. Automated static code analysis for classifying android applications using machine learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pp. 329–333, 2010. doi: 10.1109/CIS.2010.77.

Victor, H. Google kicks 60 000 apps out of its play store for low quality. April 2013. URL http://www.phonearena.com/news/Google-kicks-60-000-apps-out-of-its-Play-store-for-low-quality_id41683.

Wauters, Robin. Google: Actually, We Count Only 16,000 Apps In Android Market, Dec. 2009. URL http://techcrunch.com/2009/12/16/google-android-market/.

Zhu, Hengshu, Cao, Huanhuan, Chen, Enhong, Xiong, Hui, and Tian, Jilei. Exploiting enriched contextual information for mobile app classification. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pp. 1617–1621, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1156-4. doi: 10.1145/2396761.2398484. URL http://doi.acm.org/10.1145/2396761.2398484.