Predicting the Programming Language of Code Snippets and StackOverflow Questions Using Natural Language Processing

Abstract— **Predicting programming languages from code snippets and textual data is a challenging task. Online forums like Stack Overflow and code repositories such as GitHub rely on accurate language tags to connect the knowledge seekers to knowledge providers. In this paper, we combine ideas from the fields of Natural Language Processing and Machine Learning to predict programming languages for any data consisting of code snippet and text. Our classifier achieves an accuracy of 84% in predicting the top 24 programming languages by combining features from code snippets and textual information in Stack Overflow questions. Our results show that using Machine Learning techniques on combination of textual and code features gives better performance than using text or code features individually. Unlike other black box Machine Learning approaches, we conduct an extensive study of our feature spaces to identify the similarities and differences among programming languages. We also evaluate and compare our results with a paid API service- 'Programming Language Identification' tool by the Algorithmia market place which is shown to predict languages directly from code snippets with an accuracy of 62.8% across 20 languages.**

Index Terms— **Stack Overflow, Machine Learning, Programming Languages, Natural Language Processing and Algorithmia.**

I. INTRODUCTION

Traditionally, software projects rely on experienced programmers, code documentation and textbooks to understand the features and functionalities of a programming language. In the last decade, the advent of social media forums for programmers, such as Stack Overflow, Quora and Hacker News, have made learning and coding much simpler and faster. Young inexperienced developers, instead of waiting for experts in their projects, can rely on social forums to help debug their code or to implement some esoteric functionalities/libraries in programming languages.

Along with the growth of online social forums for programming, the complexity and the features supported by various programming languages have also increased over the last decade due to easy access to documentation and the growth of open source repositories such as GitHub, which enable easy access of source code of new libraries. In addition, there has also been an increased adaptation of esoteric languages such as Haskell and Scala, genesis of new languages based on existing languages such as Typescript (from JavaScript), Hack (from PHP), Julia (from Matlab, C and Fortan) etc. and also splitting of languages based on standards such as C++11, C++14, C++17 standards for C++, Java7, Java8, Java-EE6, Java9 standards for Java.

To help developers handle this increased complexity, forums like Stack Overflow rely on the tags of questions to match them to other experienced users who can provide answers for them. However, new users in Stack Overflow or novice developers may not tag the posts correctly. This leads to posts being down voted and flagged by moderators even though the question may be important and add value to the community. This problem can be solved if the posts are automatically tagged, especially for programming languages. This motivates the main question we address in this work: *Can we predict programming languages from code snippets and the text of StackOverflow questions?*

Existing solutions to this problem are not satisfactory. Integrated Development Environment (IDE) such as CLion, Eclipse, and text editors such as Notepad++, SublimeText, Atom, predict the language based on file extension rather than the source code itself. This can cause inconvenience to the users as they need to create the file with correct extension manually to enable syntax highlighting in these editors. In [18], the authors built a classifier to predict the programming language tags in Stack Overflow. 1000 posts were extracted to train and evaluate the model across 18 programming languages. Support Vector Machine algorithm was used to train the model. However, the method only achieves 60% accuracy and does not work well on small code snippets.

There are other tools available such as Linguist by GitHub and the Programming Language Identification (PLI)[1] tool by Algorithmia (an AI market) that can predict programming language based on code snippets [12]. It is claimed that PLI can predict 20 languages with a reported accuracy of 99.4% top1 accuracy on GitHub code snippets. However, these tools are trained on complete source code files or library files such as one from GitHub. Therefore, these tools perform better on large source code files and libraries compared to small snippets of code that are in Stack Overflow and other software forums. This leads us to the first question.

RQ 1. ***Can we predict the programming language using only code snippets embedded in a Stack Overflow question?***

In this work, we will devise a method for tag prediction that also works relatively well for short code snippets. While studying tag prediction from code snippets, we will also study how the number of lines of code impact the prediction capability of tools which predict programming languages from code snippets.

[1]In the rest of this paper, PLI stands for the Programming Language Identification tool that provided by Algorithmia.

The post description texts could be mined to further enhance our understanding of programming languages and could be useful to predict the languages without using the source code. This leads us to ask the following.

RQ 2. ***Can we predict the programming language by analyzing only the text describing a Stack Overflow question?***

Recent advances in the field of machine learning, deep learning and natural language processing [13] have resulted in an improved performance in document classification, machine translation and other predictive tasks. Therefore, machine learning and NLP techniques have immense potential in predicting the programming language from textual data. For example, it is conceivable that we can obtain a feature representation of the code snippet that can then be combined with feature representation from texts to further improve our prediction. This leads us to our final research question.

RQ 3. ***Can we improve the prediction of programming language by analyzing both the text and code snippet information inside a Stack Overflow question?***

The main contributions of our paper are:

1) A prediction model, based on Random Forest and XGBoost classifier, that predicts programming language using only the code snippet in a Stack Overflow question and obtains 61.5% accuracy score.
2) A classifier that uses only textual information embedded in Stack Overflow questions to predict the programming language and achieves an accuracy score of 81.04%, much higher than the previous best model (Baquero *et al.* [18]).
3) A third prediction method that uses a combination of code snippets and textual information available in a Stack Overflow question in order to further improve its accuracy and achieves a highest accuracy score of 84.3%.
4) A detailed study of the features used in our machine learning models through their projection into a 300 dimensional vectors space using Word2Vec [20].

II. Background and Related Work

As described in the introduction, the problem of predicting the programming language from Stack Overflow questions consisting of a text and code snippets is a topic of immense interest to the computer science research community. Despite this, our understanding of this topic is very poor.

The only previous published work on this topic, to the only best of our knowledge, is the work of Baquero *et al.* [18]. They extracted a set of 18000 question posts from Stack Overflow that contained text and code snippets, 1000 posts each, for 18 programming languages. They evaluated their classifier, trained using Support Vector Machine model, on text features and source code features. They achieved an accuracy of 60% for text features and 44% for code snippet features. Another contribution of their work is the use of feature representation to visualize the relationship between the programming languages and identify those that are closely related.

Algorithmia is a popular online market place which enables use of machine learning models as a micro service using REST API. Programming Language Identification (PLI) is one among many tools hosted by Algorithmia and currently supports 21 programming languages. The approximate cost for making 10,000 API calls to PLI is 20 USD. The tool was originally trained using source code from various GitHub repositories and claims 99.4% accuracy. Since Algorithmia is the only well known tool for predicting language from code snippets, we compare our method with Algorithmia in this work.

As related examples, some editors (Sublime, Atom) provide some interesting highlights to code on the basis of programming language. However, they require explicit mention of the extension (eg .html, .css, .py and so on). Along similar lines, Portfolio [6] is a search engine that supports programmers in finding functions that implement high level requirements in query terms. This search engine does not identify the coding language. However, it analyzes code snippets and extracts valuable functions which can be reused. Another tool developed by Holmes *et al.* [8] called Strathcona can find similar snippets of code.

To understand the factors that differentiate a good post from a bad post, Nachi *et al.* [3] studied what makes a post good or bad based on number of lines of code, quality of code and texts in a Stack Overflow post. Similarly, C. Treude *et al.* [5] analyzed Stack Overflow questions to explore the type of questions that receive a good answer. They found that the posts which contained snippets of code receive a good answer.

Software maintainers prefer to talk to each other to understand the code [9]. Researchers found three reasons why the programmers use Q&A web site: just-in-time learning, clarifying their knowledge and remembering the details. Programmers generally use the web as the main source to gather information and find a solution for their problem [4]. They use code, documentation and links to ask their questions in online forums. This gives us the opportunity to use both code and text to study about programming languages and also to predict them from these data sources. Rekha *et al.* [10] proposed a hybrid auto-system tagging system, that suggests a tag for the user who creates the question. Saha *et al.* [1] followed the approach of converting Stack Overflow questions into vectors, and then training Support Vector Machines on the basis of those vectors and eventually suggesting tags on the basis of built model. The average accuracy for tag prediction for their model was 68.47%. Although their tagging system works efficiently for some specific tags, it does not work well with some popular tags, like Java. Another suggestion by Stanley and Byrne [2] is to use a cognitively-inspired Bayesian probabilistic model. Their approach is to choose one most suitable tag for each post, the tag with highest log odds of correctness when

each tag's prior probability is known. However, their model normalizes the top activations for all posts. So it is unable to differentiate between a post where the top predicted tag is certain, and a post where the top predicted tag is questionable. This model is 65% accurate when asked to predict one tag per post on the average. In summary, most of the prior work falls short of creating a robust system which can predict programming language tags.

III. Dataset extraction and processing

In this section, we give more details about why Stack Overflow was chosen as the ground truth dataset for our study, how the data from Stack Overflow dump was extracted, and how the data was processed to answer our research questions.

A. Why did we select Stack Overflow as ground truth dataset for our proposed study?

Stack Overflow is a question and answer online forum for software developers community. Developers post questions on various coding, implementation, and debugging aspects of programming. These questions are then answered by experts in various domains. According to Quantcast [17], a popular web traffic analytics website, Stack Overflow had 2.9 Billion global visits and 7.3 Billion global views between Dec 15, 2016 and Dec 14, 2017. Stack Overflow is the most popular forum among the developers with 1.2 million unique visits to the page between Nov 23, 2016 and Dec 14, 2017.

As of July 2017, Stack Overflow had 37.21 million posts, among which 14.45 million are question posts with 50906 different tags. However, around 87% of the Stack Overflow posts are related to 150 tags. Thus, most of the Stack Overflow tags are not that popular or are relatively new. In this paper, the programming language tags in the Stack Overflow are of interest. The most popular 24 programming languages as per the 2017 Stack Overflow developer survey were selected for our analysis [22]. They constitute about 93% of questions in Stack Overflow. Even though the rest of 7% tags were also related to some programming languages, they lack a language tag. For example, Pandas is a popular library in Python for reading large Comma-Separated Values (CSV) files. The questions associated with Pandas do not have the Python tags. This could create confusion among developers who may be new to a programming language and are not familiar with all of its popular libraries. The problem of missing language tags could be addressed if posts are automatically tagged with their associated programming languages. The languages that we selected for our study are listed below:

> Assembly, C, C#, C++, CoffeeScript, Go, Groovy, Haskell, Java, JavaScript, Lua, Matlab, Objective-c, Perl, PHP, Python, R, Ruby, Scala, SQL, Swift, TypeScript, Vb.Net ,Vba

B. How did we extract and process the data from Stack Overflow Question posts?

The Stack Overflow July 2017 data dump was used for our analysis. The Stack Exchange Data dump for Stack Overflow was downloaded in .xml from archive.org. 10000 questions posted in Stack Overflow between the period of 2012 to 2016 with a score (Difference between up vote and down vote for a question post) of greater than one were selected for each programming language. This was done to ensure that these posts have been in the Stack Overflow for a long period of time and have been properly reviewed and tagged as required by the standards of the forum. The score of more than one ensured that it was a good quality post. However, we were able to extract 10,000 such posts for only 20 languages. Assembly (9121), Groovy (6978), Coffee Script (3918) and Lua (6149) had less than 10,000 posts which fitted our criterion for selection of posts. It should be noted that only the question posts which had its first tag same as the programming language of our interest was selected.

The data in .xml was parsed using xmltodict and Beautiful Soup library in python to extract the code and text from the question posts separately. A Stack Overflow question post (referred from here on as posts) consists of a title, body and embedded code. The texts from the post were separated into three categorized namely- text (title and body of the post without code fragments), the code and code and text combined. The texts from the post (title and body) were preprocessed by removing all non English characters using Regular expressions. After the preprocessing, they were subjected to dependency parsing using the Spacy Library [23]. After the dependency parsing the named entities which are nouns or noun phrases were extracted. Then, stemming and lemmatization were performed using NLTK [12] library in Python. A custom stop word dictionary was created by combining the stop words from Scikit Learn [13], Spacy and NLTK Python libraries and these stop words were removed from the text. The code snippets were split using n-gram parameter as 1 and saved separately. The processed text and code snippets were then combined to get our third dataset. Therefore, at the end of this stage, we successfully extracted the datasets given below from Stack Overflow question posts.

- 213,834 code snippets which are split with n-grams=1.
- 213,834 text snippets.
- 213,834 text snippets and code snippets combined.

In order to make our model generalize across various standards of code, we ensured that tags belonging to various language version standards were included. For example, for python, we extracted code for the tags: python-3.x, python-2.7, python-3.5, python-2.x, python-3.6, python-3.3 and python-2.6, for Java, we selected java-8 and java-7 as the tags and finally for C++, we selected c++11, c++03, c++98 and c++14 as tags. These posts were included as a part of 10000 posts selected for each language. We also ensured that our code snippets that were selected as a part of our dataset had a large variation in terms of lines of code. We plotted a box plot Fig. 1 showing the variation of lines for

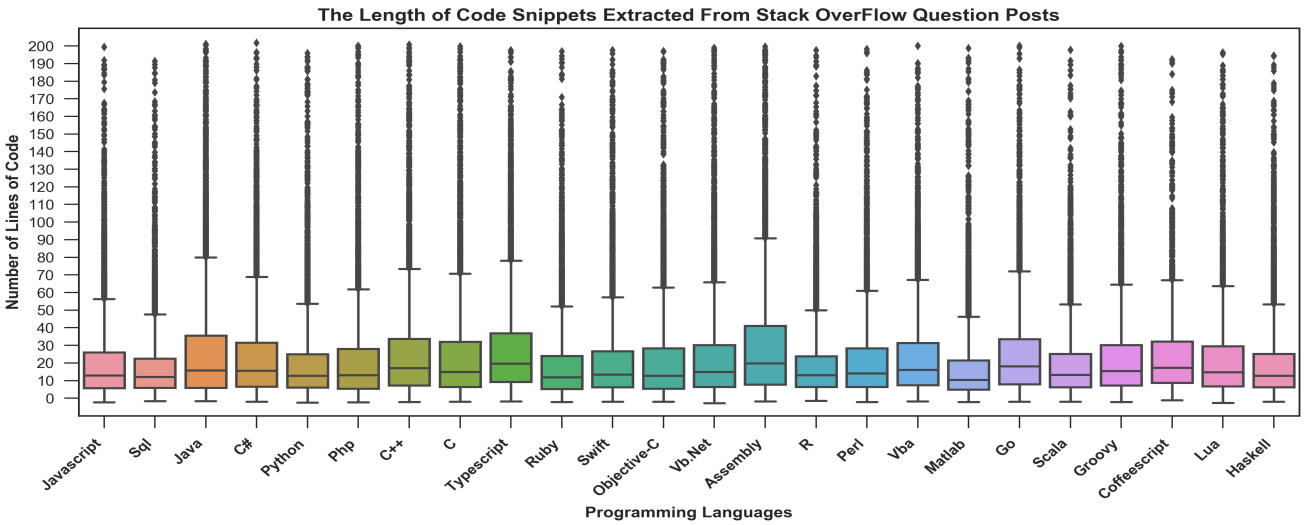


Fig. 1: The box plot showing the number of lines of code in the extracted code snippets for all the languages. It should also be noted here that there were at least 400 posts which had more than 200 lines of code, however were not included while making this plot.

the programming languages in our dataset. It may be worth noting that our dataset included code snippets as short as 6 lines of code to large snippets of 800 lines.

Evaluating and comparing the performance of our tool against Algorithmia PLI is one of the objectives of this paper. However, the 24 languages that we selected do not match with the 20 languages in Algorithmia PLI. Therefore, we extracted more data from Stack Overflow question posts for the missing languages Bash, HTML and CSS which are not present in our model but are in Algorithmia PLI, using the same process described above. These additional three languages were never included for training or testing our models. The languages supported by Algorithmia PLI are given below:

> Bash, C, C#, C++, CSS, Haskell, HTML, Java, JavaScript, Lua, Objective-C, Perl, PHP, Python, R, Ruby, Scala, SQL, Swift, Vb.Net.

IV. Methodology

In this section we discuss in detail the methodology we followed in answering our research questions.

The text and code snippets extracted from the Stack Overflow posts were split using Tf-Idf vectorizer from the Scikit learn library. we set the min-df (Minimum Document Frequency) parameter to 10 which means that we will only include words which are present in at least 10 documents among all available documents. This scheme helps us to remove the rare words from our dataset and this in turn reduces the noise. We left the max-df (Maximum Document Frequency) parameter to default since we have already removed the stop words in the data pre-processing step in section 3. We selected Random Forest Classifier and XGBoost which is a gradient boosting algorithm as our machine learning models. We report precision, recall, accuracy, F1 score and confusion matrix as our metrics for all our classifiers in Section V.

The machine learning models were hyper-tuned using RandomSearchCV - a tool for parameter search in Scikit-learn. XGboost algorithm has many hyper parameters such as minimum child weight, max depth, L1 and L2 regularization, and various evaluation metrics such as Receiver Operating Characteristic (ROC), accuracy and F1 score. Similarly, Random Forest Classifier which is a bagging classifier has a parameter called 'number of estimators' which are subtrees used to fit the model. It therefore become really important to tune the model by varying these parameters. However, parameter tuning is computationally expensive using standard Grid Search. Therefore we adopted a technique borrowed from deep learning called Random Search hyper tuning to train our models. The parameters for all our models were fixed after performing RandomSearch on the cross validation sets (Stratified ten fold cross validation). It may be noted that the dataset was split into train and test in the ratio 80:20, where train was used for training and cross validation and test set was used to report all final metrics in this paper. The code snippets, text and code snippets + text dataset models all followed the same approach described here.

Our trained model was compared with Algorithmia PLI. The evaluation process was divided into two separate steps:

1) Evaluation for the 20 languages supported by Algorithmia PLI.
2) Evaluation for 17 languages common to our model and Algorithmia PLI.

We selected 75 code snippets randomly sampled from our test dataset to perform evaluation. We used the urlib library in Python to make the API calls to Algorithmia PLI to get predictions for all 20 programming languages supported

Programming Language	Precision	Recall	F1-score
PHP	0.89	0.72	0.79
Go	0.9	0.7	0.79
Coffeescript	0.88	0.72	0.79
Groovy	0.86	0.71	0.78
Javascript	0.91	0.67	0.77
C	0.85	0.7	0.77
Haskell	0.87	0.68	0.76
Swift	0.89	0.63	0.74
Assembly	0.83	0.64	0.73
Vba	0.83	0.66	0.73
C++	0.84	0.62	0.71
Matlab	0.83	0.59	0.69
Vb.net	0.84	0.57	0.68
Scala	0.69	0.66	0.68
Lua	0.83	0.57	0.68
Python	0.85	0.53	0.65
Perl	0.77	0.57	0.65
R	0.72	0.53	0.61
C#	0.75	0.5	0.6
SQL	0.63	0.54	0.59
Ruby	0.62	0.56	0.59
Java	0.64	0.51	0.57
Typescript	0.65	0.48	0.55
Objective-c	0.15	0.86	0.25

TABLE I: Performance Metrics for our classifier trained on code snippet features. This classifier achieves an accuracy of 62.4% in predicting 24 languages from code snippets.

by it. The API call returns a JSON file with languages as key and corresponding probability for all the 20 languages. We compared the predicted language with the ground truth to generate all our metrics. Our metrics included accuracy, precision, recall and F1 score. We also created a confusion matrix to further evaluate our results. Algorithmia PLI tool and its API is a paid service and this constrained us from selecting more than 75 snippets for evaluating it.

Another important contribution of this paper is the study of the vocabulary and feature space used in our algorithms. A word2vec model [20] was trained on our extracted code and text datasets using Gensim—a Python framework for vector space modeling [19]. The resulting model represented each word in our vocabulary in a 300 dimensional vector space. However, it is impossible to visualize concepts in such high dimensional space. Therefore, we used t-SNE [21] which is a popular dimensionality reduction algorithm to reduce the number of dimensions of our word vectors to 2. We then selected top 3% words from our vector embeddings for each programming language and analyzed the word similarity and cosine distance. The code and text features which are closer to one another in vector space were selected for every language using cosine as measurement. These features were then combined across all languages (Dictionary of named entities) and used in our machine learning models. These results are reported in Section V.

V. Results and Discussions

In the first part of this section, we describe in detail the results we have obtained towards answering the following research question. All the results reported under this section were obtained using the test dataset (Total data split as 80% train and 20% test).

RQ 1. *Can we predict the programming language using only code snippets embedded in a Stack Overflow question?*

We trained two machine learning classifiers on the code snippets extracted from Stack Overflow questions. The Random Forest and XGBoost classifier achieved an accuracy of 60.4 and 62.4% respectively. These models were trained on the code features extracted from the code snippets using Word2Vec as described in the methodology section. In this section, we will focus only on the results of XGBoost classifier which is the top performing model. Languages such as Groovy (0.78), Go (0.79), Coffee Script (0.79) and PHP (0.79) had the highest F1 Score metrics as indicated in brackets. SQL (0.59), Java (0.57), Type Script (0.55) and ruby (0.59) performed poorly. However, the worst performing programming language was Objective-C with an F1 score of 0.25 and high recall of 0.86 but with poor precision of 0.15. This means that the classifier confuses Objective-C with other languages. When we analyzed the code features for Objective-C, we found that it shares code syntax with multiple other languages such as C, Java, Python, C++ and even C#. There are not many unique code features available to Objective-C which could enable the classifier to uniquely identify it. A small number of code features that we extracted for Java and SQL can be found in Fig. 4 and Fig. 5 respectively. The detailed performance metrics of our code feature based classifier can be found in Table I.

The next step was to compare our code snippets based classifier with Algorithmia PLI classifier. We performed this comparison by evaluating the results we obtained by running the prediction of programming languages using the tool from Algorithmia PLI for 1500 snippets of code across 20 different programming languages. The tool gave an accuracy of 62.8% in predicting languages from code snippets with an F1 score of 0.64. The tool has a precision of 0.68 and a recall of 0.63. The confusion matrix for this prediction task is shown in Fig. 2. We can infer from the confusion matrix that some languages such as Scala, Haskell, Ruby, Objective-C and Python perform much better compared to languages such as HTML, CSS, JavaScript. SQL performs the worst. Scala and Haskell are not often confused with other languages due their unique syntax. It is bit surprising to us that Objective-C is not confused with C language since they both have very similar syntax. It could be because Objective-C is much more similar to C++ than C language. This is evident from some Objective-C code snippets which are confused with languages such as Java, Ruby, Scala and C which is similar to the results from our code snippet based classifier.

HTML, CSS and JavaScript are used together in web development and moreover, CSS and JavaScript are often embedded in the HTML files which could explain why the tool performed very poorly in identifying these languages. We manually analyzed the code snippets from HTML, CSS

Programming Language	Precision	Recall	F1-score
Objective-c	0.95	0.79	0.86
Haskell	0.72	0.8	0.76
Vb.net	0.94	0.63	0.75
Scala	0.68	0.84	0.75
PHP	0.84	0.65	0.74
Swift	0.82	0.68	0.74
Perl	0.8	0.65	0.72
Python	0.67	0.76	0.71
Bash	0.81	0.61	0.7
R	0.85	0.59	0.69
C++	0.6	0.71	0.65
C#	0.62	0.63	0.62
Java	0.75	0.51	0.6
C	0.61	0.57	0.59
SQL	0.81	0.45	0.58
Lua	0.38	0.71	0.49
Javascript	0.45	0.49	0.47
Ruby	0.34	0.76	0.47
CSS	0.61	0.33	0.43
HTML	0.39	0.4	0.4

TABLE II: Performance Metrics for Algorithmia PLI on code snippetS features. This classifier achieves an accuracy of 62.8% in predicting 20 languages from code snippets.

and JavaScript and found that in more than 60% of the cases, the tool generated wrong predictions because of them being used together in the same code snippet. However, in rest of the cases the tool simply got them wrong. We speculate that this could be due to the reason that training data might have had embedded CSS and JavaScript in them which makes the tool perform poorly on these languages compared to other languages. The poor performance on SQL could be attributed to the fact that SQL is mostly embedded in the code snippets of other programming languages such as Python, C, C++, Java etc. From the confusion matrix in Fig 2, we can see that SQL is mostly confused with Scala, Ruby and R. The detailed performance metrics of Algorithmia PLI tool is reported in Table II.

RQ 2. *Can we predict the programming language by analyzing only the text describing a Stack Overflow question?*

We trained two machine learning models (Random Forest and XGBoost) on text features extracted from Stack Overflow question. Random Forest and XGBoost achieved an accuracy of 78.4% and 81.04% respectively. Compared to the models trained on the code features, this is a significant improvement in performance. The top performing languages based on F1 score metrics are Swift (0.94), Go (0.94), Coffee Script (0.93), C (0.93), Haskell (0.92), Groovy (0.91) Assembly (0.91) and Javascript (0.91). It should be noted that Objective-C performs really good when trained on text with an F1 Score of 0.90 as compared to an F1 score of 0.25 using code features. The worst performing languages are Java (0.55), C# (0.59) and SQL (0.60). This is consistent with their performance with code features. This also implies there is not sufficient information available in the feature space (both code and text) to effectively create a good decision

Programming Language	Precision	Recall	F1-score
C	0.97	0.94	0.96
Swift	0.99	0.92	0.95
Go	0.96	0.93	0.95
Coffeescript	0.97	0.93	0.95
Groovy	0.98	0.9	0.94
Assembly	0.94	0.91	0.93
Haskell	0.94	0.92	0.93
Objective-c	0.92	0.92	0.92
Javascript	0.91	0.91	0.9
C++	0.96	0.87	0.91
PHP	0.93	0.85	0.89
Matlab	0.9	0.87	0.88
Perl	0.87	0.84	0.86
Python	0.91	0.8	0.85
Vb.net	0.82	0.89	0.85
Scala	0.8	0.91	0.85
Vba	0.82	0.84	0.83
R	0.81	0.82	0.81
Typescript	0.82	0.77	0.8
Ruby	0.74	0.8	0.77
Lua	0.77	0.68	0.72
SQL	0.6	0.72	0.65
C#	0.7	0.61	0.65
Java	0.52	0.69	0.6

TABLE III: Performance Metrics for our classifier trained on code and text features. This classifier achieves an accuracy of 84.2% in predicting 24 languages.

boundary for these languages. It should be also noted that languages like Go, Coffee Script, Haskell, Groovy, Swift and PHP performed equally well using code and text features.

On analysis of the feature space of top performing languages, we found that these languages have unique code features (key words/identifiers) and text features (libraries, functions). For example when we visualize the text based features for Haskell, we see words like 'GHC', 'GHCI', 'Yesod' and 'Monad'. 'GHC', 'GHCI', are compilers for Haskell, 'Yesod' is a web based framework and 'monad' is a functional programming paradigm (Haskell is a functional programming language). Most of top performing languages also do not have a very big feature space (Vocabulary) as compared to more popular languages like Java, Python, C# etc which has large number of libraries, standard functions and supports multiple programming paradigms which increases the size of feature space. A large feature space adds more complexity to the machine learning models. A small number of text features that we extracted for Java and SQL could be found in Fig. 4 and Fig. 5 respectively.

RQ 3. *Can we improve the prediction of programming language by analyzing both the text and code snippet information inside a Stack Overflow question?*

We combined both our text features and code features extracted from the Stack Overflow questions to train Random Forest and XGBoost classifiers. XGBoost classifier achieved an accuracy of 84.2% where as Random Forest achieved 81.9% accuracy on the test data. We can see that this is less than 3% improvement over our text feature based dataset. The detailed performance metrics of our code + text

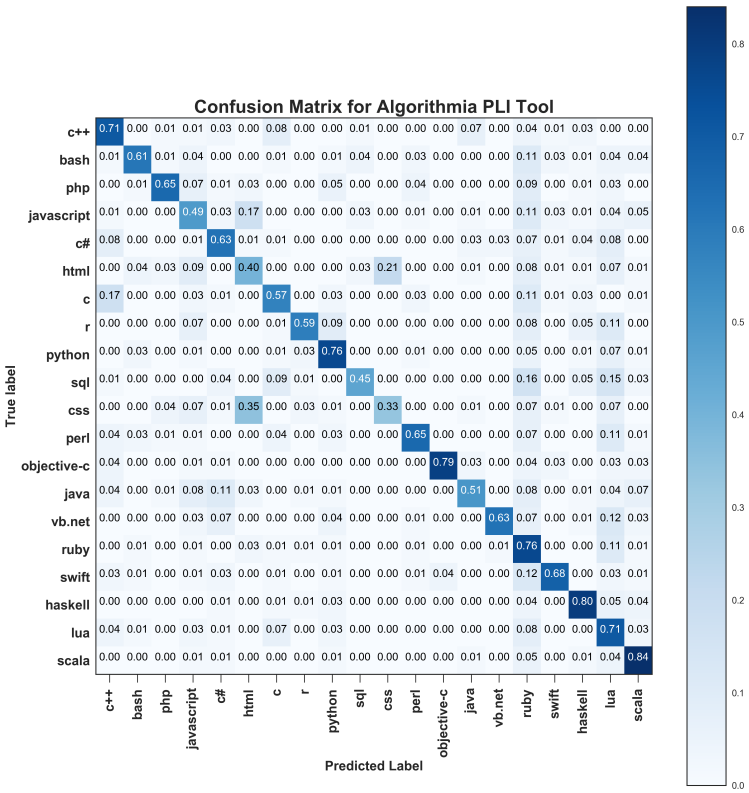


Fig. 2: Confusion matrix for Algorithmia PLI. We can see that there is a very high degree of confusion among the languages. This tool achieved a F1 score of 0.64.

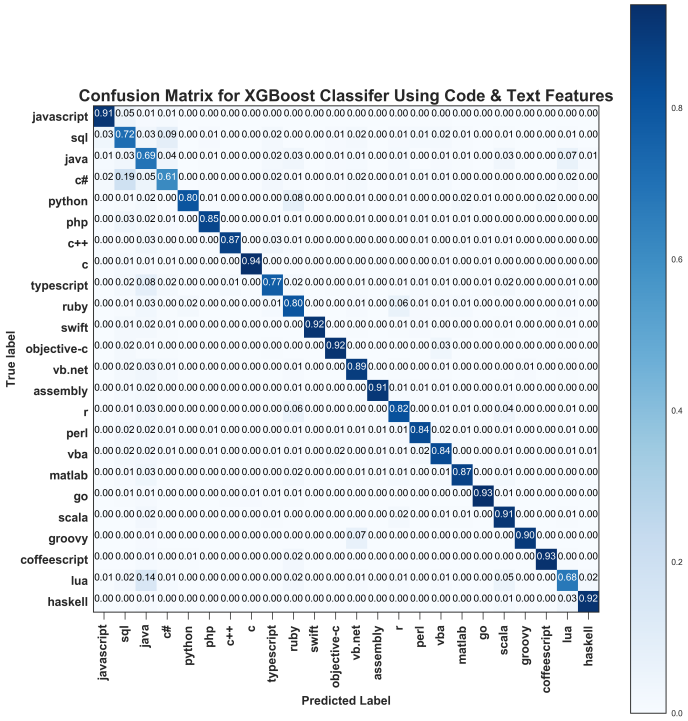
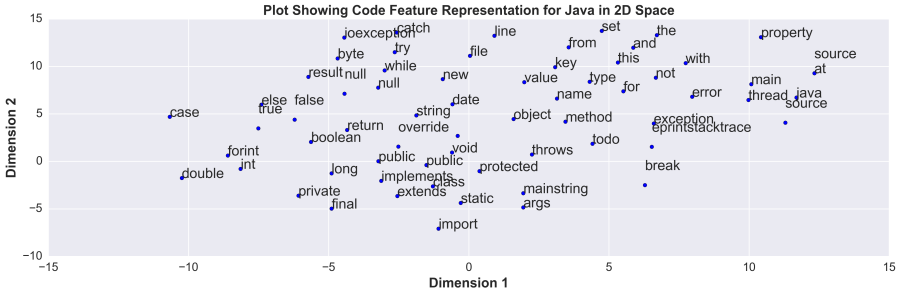
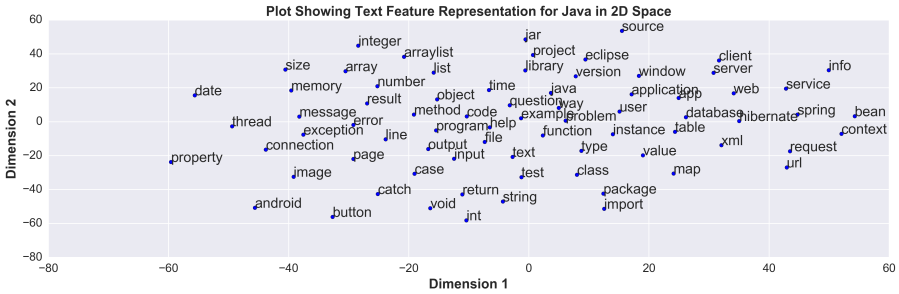


Fig. 3: Confusion matrix for the XGboost classifier trained on code and text features. The diagonal of the matrix is very prominent and the degree of confusion is very low as compared to confusion matrix for Algorithmia PLI.

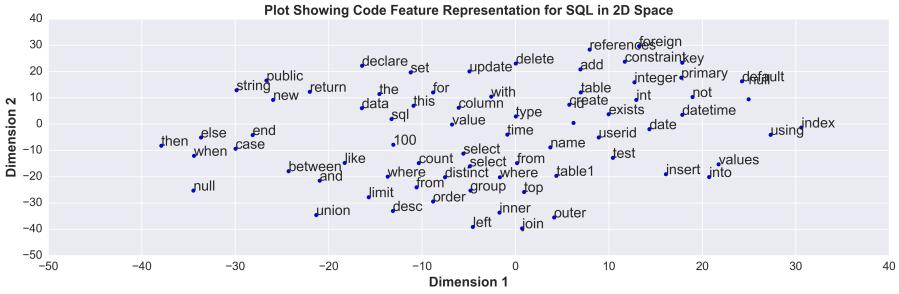


(a) Code Features of Java

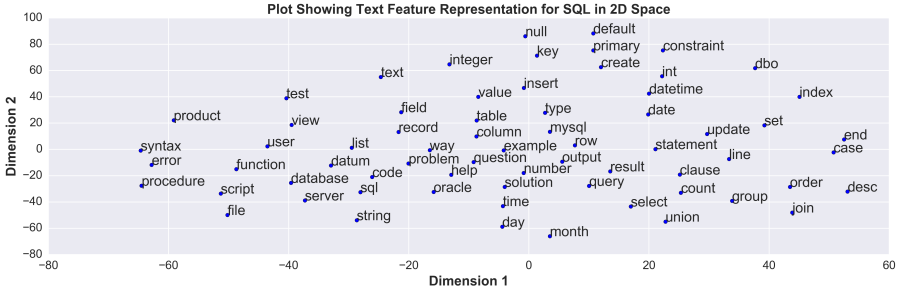


(b) Text Features of Java

Fig. 4: Plot showing code and text features of Java represented in 2 dimension after using t-SNE on a trained Word2Vec model.



(a) Code Features of SQL



(b) Text Features of SQL

Fig. 5: Plot showing code and text features of SQL represented in 2 dimension after using t-SNE on a trained Word2Vec model..

combined feature classifier could be found in Table III. Also the confusion matrix for our top performing model could be found in Fig. 3. The performance of Java, C and C# improved by only 5% by combining code and text features. There was a significant improvement in F1 Score of Objective-C (0.92) compared to only a score of 0.25 using code features alone. Almost all the languages except Lua, SQL, C# and Java had both high precision and recall indicating that our machine learning model is more robust. Previous works including Baquero *et al.* [18],Rekha *et al.* [10] and Saha *et al.* [1] all reported similar results for these languages. However, our results for these languages were significantly better both in terms of precision and recall. Combining commonly used code features (identifiers, commonly used functions etc.)

Model	Description	Accuracy	Precision	Recall	F1-score
Previous work					
Baquero [18], code snippet	A model trained using Support Vector Machine from 18000 question posts from Stack Overflow using code features.	44.6%	0.45	0.44	0.44
Baquero [18], text	A model trained using Support Vector Machine from 18000 question posts from Stack Overflow using text features	60.8%	0.68	0.60	0.60
Our experiments:					
Algorithmia PLI- 20 lang.	Evaluated for 20 languages- Excluded markdown	62.8%	0.68	0.62	0.63
Algorithmia PLI- 17 lang.	Evaluated for 17 languages -Excluded Bash, HTML, CSS	66.9%	0.72	0.66	0.67
Our classifier: code features	XGBoost classifier trained on 17100 Stack Overflow Question posts using 5000 code features	62.4%	0.76	0.62	0.67
Our classifier: text features	XGBoost classifier trained on 17100 Stack Overflow Question posts using 5000 text features	81.0%	0.82	0.81	0.81
Our classifier: code and text features	XGBoost classifier trained on 17100 Stack Overflow Question posts using 10000 features	84.2%	0.85	0.84	0.84

TABLE IV: Table showing comparison between previous works and our classifiers.

and text features (library names, compiler names etc.) for languages helped us to train better machine learning models.

The Word2Vec representations shown in Fig. 4 and Fig. 5 were created by splitting the code and text snippets and training their vector representations separately for each programming language. There were more than 200 features extracted using Word2Vec for each language but we have shown only around 70 features in our figures because including a large number of features in our figures can make it obscure and unreadable. In a vector space, features which are used in the same context lies closer in the high dimensional space. For example, in the Java code Fig. 4, we can see that 'public','class','implements','extends' are all close to one another in vector space since they are mostly used together in a line of code. Another example we can look at is for SQL code Fig. 5 where features like 'foreign', 'primary', 'key', 'constraint' and 'reference' are all used in the same context of setting relationship between tables.

The smaller cosine distance between features does not mean they are used in the same line of code but used in the same context (Declaring Relationships in tables for example). Now when we look at the text features for Java we can see that this include some code features as well. This is because in Stack Overflow, people describe their issues by including code based identifiers and features along with the text not just in code blocks. However, we can also see 'Netbeans', 'Eclipse' which are popular IDE's for Java also present. Another key observation from the Java text feature is that 'Android' and 'spring'- a framework for web and desktop applications are located at opposite sides of the plot(maximum cosine distance). This means that an Android developer might never use spring framework for his developmental activities. At the same time, 'spring' is located close to 'web', 'service', 'url', 'request', 'xml' etc which indicates that Spring framework is used alot in web development in Java.

The application of Word2Vec is not limited to classification tasks. Training these models on both code and text features could unravel valuable insights about programming languages. This method is more robust than other methods such as Latent Dirichlet Allocation (LDA) which are still popular among the software engineering community. The impressive results that we achieved in this paper could be attributed to the fact that we didn't follow a black box approach towards machine learning. Careful analysis and visualization of features contributed more to our trained model that just machine learning. Using dependency parsing and extracting named entities using Neural Network techniques through Spacy helped us to reduce noise and extract import features from Stack Overflow. This made a drastic improvement in our performance compared to previous works. We have summarized our results in Table IV.

VI. Future Work

The study of programming language prediction from text and code snippets is still new and a lot remains to be done. Most of the existing tools focus on file extensions rather than the code itself for predicting the language. We evaluated the most popular tool Algorithmia PLI and found it insufficient in predicting the languages. In the recent years, there has been tremendous progress made in the field of Deep Learning, especially time series or sequence based models such as Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) Networks. These networks can be given a sequence of text basically character by character as input and could predict the category of the sequence. We could give the source code one character at a time as input to these RNN and LSTM models and we would be able to predict the target programming language. One small draw back of this methodology is its high computational cost and it might need large Graphics Processing Unit (GPU) clusters to train and test.

Natural Language Processing and Machine Learning techniques performed much better in predicting languages compared to tools predict in from source code. Therefore, we need to ensure that additional research is carried out to improve the results of this task. Stack Overflow texts are pretty unique in the sense they capture both the tone, sentiments and vocabulary of the developer community. There might be a variation between this vocabulary for each programming language. Therefore, it is important that we capture, understand and separate the vocabulary for each

programming language. We also recommend training a Convolutional Neural Network (CNN) which are used along with Word2Vec for various Natural Language Processing (NLP) related tasks. CNN have shown considerable improvement in document classification tasks compared to most simple classifiers.

Our current model was trained and evaluated only on Stack Overflow question posts. In the future it could be interesting to evaluate the model on programming blog posts, library documentation and bug repositories. This would help us understand the generalization of our trained models.

VII. Threats to Validity

Construct Validity: In the process of creation of our data from Stack Overflow Question posts, we selected only programming language tags. However, some tags synonymous for the languages were not included in our data extraction process. For example, 'SQL SERVER', 'PLSQL', 'MICROSOFT SQL SERVER' etc are related to SQL programming language but were not included in our dataset.

Internal validity: When we extracted our dataset we used dependency parsing to selected the named entities and thereby include only the most relevant code and text features. The use of dependency parsing could result in loss of important vocabulary which might affects the results of our experiments with text and code features. However, we have manually analyzed the vocabulary before and after the dependency parsing and other text processing steps to ensure that information related to the languages are not lost. Also, selecting features such as lines of code, programming paradigm etc. as a part of the model could have improved the results but was not included as a part of our methodology.

External validity: We have only used Stack Overflow as the source of data for our analysis. We have not explored other sources such as GitHub repositories. Therefore, we cannot be absolutely confident that our results would be the same across all the sources of code snippets and texts/documents on programming languages. Our research has only focused on one tool, Algorithmia, for answering our research questions. This is mainly due to the lack of availability of open source tools for predicting languages. We have tried our best to include almost all the most commonly used programming languages for our study. However, there are still more popular and powerful languages such as Cobol, Go, pascal etc which were not considered for this study.

VIII. CONCLUSIONS

In this paper we discussed the importance of predicting languages from code snippets and textual data. We argued that the existing tools for predicting the programming language from code snippets are not adequate enough considering the challenges and complexity of today's programming languages. We evaluated a paid API service called ProgrammingLanguageIdentification by Algorithmia market place and found that it achieved only an accuracy of 62.8% on code snippets from Stack Overflow posts. This is in direct contrast with the claimed accuracy of the tool which is 99.4%. On analysis of the confusion matrix, we observed that some languages such as Objective-C, Haskell etc. can be predicted better compared to languages such as HTML, CSS and JavaScript which are often used together. SQL is often used as embedded code in other languages such as C, Java, C++ and python and this also affects the accuracy of the Algorithmia PLI which predicts language from source code.

Our main contribution is a method that achieves an impressive accuracy of 84.2% by combining the code and textual features from the posts in Stack Overflow posts to predict programming languages. In contrast, we observed that using code snippet features only gives an accuracy of 62.4% while using the textual features only provides an accuracy of 81.04%. The use of techniques such as extraction of named entities using dependency parsing, selection of feature space using word2vec and selection of simple and highly regularized machine learning models have helped us achieve this state-of-the-art accuracy.

References

[1] A. K. Saha, R. K. Saha, and K. A. Schneider, A discriminative model approach for suggesting tags automatically for stack overflow questions, in Proceedings of the International Workshop on Mining Software Repositories. IEEE Press, 2013, pp. 7376.

[2] C. Stanley and M. D. Byrne. "Predicting Tags for StackOverflow Posts". In Proceedings of ICCM 2013 (12th International Conference on Cognitive Modeling), 2013.

[3] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns,"What makes a good code example? A study of programming Q&A in StackOverflow", in Proc. ICSM, pages 2535, 2012.

[4] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code", in Proceedings of CHI 2009, New York, NY, USA, 2009, pp. 15891598.

[5] C. Treude, O. Barzilay, and M.-A. Storey, "How Do Programmers Ask and Answer Questions on the Web?" in Proceedings of ICSE 2011, New York, NY, USA, 2011, pp. 804807.

[6] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: A Search Engine for Finding Functions and Their Usages", in Proceedings of ICSE 2011, 2011, pp. 10431045.

[7] M. Revelle, B. Dit, and D. Poshyvanyk, "Using Data Fusion and Web Mining to Support Feature Location in Software", in Proceedings of ICPC 2010, 2010, pp. 1423.

[8] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona Example Recommendation Tool", in ACM SIGSOFT Software Engineering Notes, New York, NY, USA, 2005, pp. 237240.

[9] B. Seaman, "The Information Gathering Strategies of Software Maintainers", in Proceedings of ICSM 2002, 2002, pp. 141 149.

[10] V. S. Rekha, N. Divya, and P. S. Bagavathi. "A hybrid auto-tagging system for stackoverflow forum questions". In Proceedings of the 2014 International Conference on Interdisciplinary Advances in Applied Computing, ICONIAAC 14, pages 56:156:5, New York, NY, USA, 2014. ACM.

[11] Algorithmia, "ProgrammingLanguageIdentification tool", 2017. [Online]. Available: https://www.algorithmia.com.

[12] E. Loper, S. Bird, "NLTK: The natural language toolkit", in Proc. Interact. Present. Sessions Association for Computational Linguistics, 2006, pp. 6972.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", Journal of Machine Learning Research, 2011.

[14] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design Lessons from the Fastest Q&A Site in the West", in Proceedings of the 2011 annual conference on Human factors in computing systems, New York, NY, USA, 2011, pp. 28572866.

[15] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider. "Answering questions about unanswered questions of stack overflow". In MSR, 2013.
[16] SoStats, 2017. [Online]. Available: https://sostats.github.io/last30days/
[17] Quantcast, 2017. [Online]. Available: https://www.quantcast.com
[18] J. Baquero, J. Camargo, F. Restrepo-Calle, J. Aponte, and F. Gonza lez, "Predicting the Programming Language: Extracting Knowledge from Stack Overflow Posts". In Advances in Computing, 2017.
[19] R. Rehurek and P. Sojka, "Gensimpython framework for vector space modelling", NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 2011.
[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. "Distributed representations of words and phrases and their compositionality". In NIPS, pages 3111, 2013.
[21] L.J.P. van der Maaten and G.E. Hinton. "Visualizing data using t-SNE". Journal of Machine Learning Research, 9(Nov):24312456, 2008.
[22] Developer Survey Results 2017. [Online]. Available: https://insights.stackoverflow.com/survey/2017#technology
[23] M. Honnibal and M. Johnson, "An improved non-monotonic transition system for dependency parsing", in Proc. of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2015, pp. 13731378.