QuIET: A text classification technique using automatically generated span queries

Vassilis Polychronopoulos[1]
University of California, Santa Cruz
vassilis@soe.ucsc.edu

Nick Pendar[2]
Skytree, Inc.
nick.pendar@skytree.net

Shawn R. Jeffery
Groupon, Inc.
sjeffery@groupon.com

***Abstract*—We propose a novel algorithm, QuIET, for binary classification of texts. The method automatically generates a set of span queries from a set of annotated documents and uses the query set to categorize unlabeled texts. QuIET generates models that are human understandable. We describe the method and evaluate it empirically against Support Vector Machines, demonstrating a comparable performance for a known curated dataset and a superior performance for some categories of noisy local businesses data. We also describe an active learning approach that is applicable to QuIET and can boost its performance.**

I. Introduction

Categorization and labeling of textual data by applying sets of multi-term queries that denote particular categories is common practice in the industry. Given a set of categories, domain experts write a set of queries for each category. Every document is labeled as belonging to a specific category based on the set of queries that hit it.

Text categorization systems based on query generation are traditionally produced manually and are therefore costly. The cost of manual query set construction is even higher for complex categorization taxonomies involving large sets of texts. Traditional machine learning techniques are increasingly used as an alternative to query-based text categorization [1]. The models produced by machine learning techniques are not human understandable and require expertise from their users. Also, text datasets usually suffer from high category skew which poses a challenge to machine learning methods. Datasets of documents that contain noise and spam are common, especially if the data are obtained online.

As a motivating example, Groupon maintains a database of millions of local businesses. Each local business needs to be tagged with the services that it provides based on a Groupon-specific taxonomy of services describing local businesses. We want a method that can perform the tagging automatically, as it is costly to manually go over all local businesses data. The training sets we obtain through crawling of merchant webpages and crowdsourcing are particularly noisy. In this application, low performance comes at a cost because sales staff waste time to engage with the wrongly classified merchants, which means we need additional manual labor on top of the automated text categorization. We desire a text categorization method that can mitigate the effects of noise in the training set and can return human understandable models that can be validated and fine-tuned by their users.

We describe a new method that builds binary text classifiers based on automated query generation. The method is called QuIET, which stands for Query Induction and Extraction from Text. Each classifier determines membership or non-membership in a target category. We can use multiple binary classifiers to tag unlabeled documents with all the categories to which they belong. QuIET extracts features from a set of labeled documents that act as the training corpus and uses them to generate a set of scored queries. It then uses the queries and their associated scores to predict the category of unlabeled texts. The query sets are human understandable and achieve high performance even with the use of noisy training data with high category skew. Manual annotation of a training set is far less costly than manually constructing category-characteristic sets of queries. The method is designed to scale computationally to large datasets, as its construction time is linear to the size of the training corpus and it employs methods that can efficiently determine the queries that hit an unlabeled document during classification.

In this paper, we describe QuIET in detail, and provide empirical evaluation against Support Vector Machines. QuIET achieves comparable performance in the case of the Modapte split of the Reuters-21578 dataset and a superior performance for noisy local businesses website data. We also implement active learning using QuIET, and show that it boosts its performance by isolating noise and outliers in the training set.

II. Related Work

Numerous machine learning techniques for text categorization have been studied in the literature, including Decision Trees [2], Naive Bayes [3], Support Vector Machines [4] and boosting methods [5]. Recently, Tripathi et al. [6] proposed a hybrid method for text categorization involving various two-classifier and four-classifier combinations.

Support Vector Machines (SVM) achieve the highest efficiency in many cases. To train an SVM classifier for text categorization, we need to extract informative features from the training corpus to build a feature set. Typically, features are words in the document corpus (unigrams). Each document is described as a feature vector based on the feature words it contains, and the SVM model is trained using the feature vectors in the same manner as in non-textual applications.

[1] Contributed to this work during an internship at Groupon
[2] Formerly at Groupon

Studies have suggested the use of n-grams as features for SVM, but these approaches generally suffer from the curse of dimensionality.

In [7], Hirsch et al. described a text categorization method that automatically generates Apache Lucene queries and uses them to classify texts. Human comprehensibility of the produced query sets provides the advantage that the results can be validated and fine-tuned by humans. Query sets obtained with this method can also be used for different tasks such as information extraction. QuIET's automatically generated queries are not AND, OR and NOT queries as in [7], but multi-term span queries. Moreover, QuIET uses a sophisticated separation metric to assign scores to the generated queries. The scores form an integral part of the method for classifying unseen texts.

III. Preliminaries

A. Queries

Definition Let D be a corpus of training documents. A *feature* from set D is a word that occurs in documents in D and is selected to be a member of a feature set F.

Definition *Term array* t is an ordered list of words. Let t^n be a term array of length n (with $n = 0$ defining the empty array), and t^n_i, where $i \leq n$, be the ith element in term array t^n. The length of term array t^n is $|t| = n$. Any sequence of tokens corresponds to a term array and vice versa.

Definition Given a text segment ts with n tokens we define a *subsequence yielding progression*, denoted SYP, to be a strictly increasing integer function $p : \{1, .., m\} \rightarrow \{1, ..., n\}$ with $m \leq n$. Note that every SYP corresponds to a subsequence of tokens from ts, by substituting each number $p(i)$ in the sequence $p(1)...p(m)$ with the token in position $p(i)$ of ts. A given subsequence may be the result of more than one SYPs.

Example For text segment ts="very healthy and very tasty meal', SYP p_1, where $p_1(1) = 2, p_1(2) = 3, p_1(3) = 5$ corresponds to subsequence "healthy and tasty" . SYP p_2, where $p_2(1) = 1, p_2(2) = 5$ corresponds to subsequence "very tasty" as is SYP p_3, where $p_3(1) = 4, p_3(2) = 5$.

Definition *Query* q is a tuple $\langle t^w, g\rangle$, where t^w is a term array and $g \in \mathbb{N}$ denotes the maximum gap allowed between the token indices of the first and last terms in t^w in a given text segment. For all queries of term arrays t^1, i.e. single term queries, we vacuously define $g = 0$. For a query with term array t^w with $w > 1$, it must that $g \geq w - 1$. For ease of reference, let $\mathrm{terms}(q_i)$ and $\mathrm{gap}(q_i)$ refer to t^w and g components in query q_i, respectively. Let Q be the set of all queries.

Definition With respect to a specific corpus D we define function $\mathrm{hit} : Q \rightarrow \mathcal{P}(D)$ be a relation from the set of queries into the powerset of the corpus documents. We say that query $q : \langle t^w, g\rangle$ hits document d if there exists a text segment ts in d with n tokens for which there is a SYP $p : \{1, ..., w\} \rightarrow \{1, ..., n\}$ which corresponds to the sequence of $\mathrm{terms}(q)$ and $\mathrm{gap}(q) \geq p(w) - p(1)$

Example Assume D is a set of two documents D_1 ="*come enjoy our delicious sandwiches"* and D_2 ="*Join us for a soothing fish pedicure"* . We derive a set of features F from D, where $F = \{$delicious, enjoy, join, pedicure, sandwiches, soothing$\}$. An example of a t^2 term array is [*enjoy,sandwiches*] and an example of t^3 term array is [*join,soothing,pedicure*]. Let $q_1 = \langle$[enjoy,sandwiches]$, 3\rangle$, then $hit(q_1, D) = \{D_1\}$. For $q_2 = \langle$[enjoy,sandwiches]$, 1\rangle$, $hit(q_2, D) = \emptyset$, because the terms *enjoy* and *sandwiches* do not occur in any documents in D within a gap of 1 or less.

Definition *Query subsumption* denoted by $\sqsupseteq_q$, defines a partial order over Q; $q_i \sqsupseteq_q q_j$ if for every set of documents D, $\mathrm{hit}(q_i) \supseteq \mathrm{hit}(q_j)$. We say that query q_i *subsumes* query q_j. It follows from the subset relation that query subsumption is reflexive and transitive.

Example Query $\langle$[*sandwiches*]$, 0\rangle$ subsumes query $\langle$[*enjoy, sandwiches*]$, 3\rangle$ which is in turn subsumed by query $\langle$[*enjoy, sandwiches*]$, 5\rangle$.

Definition *Term subsumption* denoted by $\supseteq_t$, defines a partial order over the set of all term arrays. For two term arrays t^w, u^q, $t^w \supseteq_t u^q$, that is, t^w term subsumes u^q, if the sequence defined by the terms in u^q contains at least one subsequence which contains all the terms of t^w in order. Term subsumption is reflexive and transitive.

Example For term arrays $r^2 =$ [*enjoy, sandwiches*] and $r^3 =$ [*enjoy, delicious, sandwiches*], $r^2 \supseteq_t r^3$

Definition Let t^w, u^q be two term arrays, and $t^w \supseteq_t u^q$. Because of the term subsumption, there exists at least one subsequence of the sequence defined by term array u^q that contains all the terms of t^w in order, and thus, there is at least one SYP over the text defined by the sequence of u^q that corresponds to the sequence of t^w. We call any such progression over u^q a *critical progression*. We define the m-progression $\mathrm{m} : \{1, ..., w\} \rightarrow \{1, ..., q\}$ to be the critical progression such that

- The span of the largest and smaller number of the m-progression, that is, $\mathrm{m}(w) - \mathrm{m}(1)$ is the minimum among all critical progressions.
- $m(1)$ is the leftmost index of any critical progression of the same span, that is, $m(1)$ is less or equal to $p_i(1)$ for all critical progressions p_i for which $\mathrm{p_i}(w) - \mathrm{p_i}(1) = \mathrm{m}(w) - \mathrm{m}(1)$

We define:

- The *left outer omission* of t^w and u^q, denoted $\mathrm{lom}(t^w, u^q)$, is the term array $(u^q_1, ..., u^q_{\mathrm{m}(1)-1})$. For $\mathrm{m}(1) = 1$, the left outer omission is the empty term array.
- The *right outer omission* of t^w and u^q, denoted $\mathrm{rom}(t^w, u^q)$, is the term array $(u^q_{\mathrm{m}(w)+1}, ..., u^q_q)$. For $\mathrm{m}(w) = q$, the right outer omission is the empty term array.
- The *outer omission* of t^w and u^q, denoted $\mathrm{oo}(t^w, u^q)$ is the term array that results from the concatenation of term

arrays $\mathrm{lom}(t^w, u^q)$ and $\mathrm{rom}(t^w, u^q)$

As an example, consider term arrays: $t = \langle b, e\rangle$ and $u = \langle a, b, c, d, e, f\rangle$, for which $t \sqsupseteq_t u$. We have: $lom(t, u) = \langle a\rangle$, $rom(t, u) = \langle f\rangle$ and $\mathrm{oo}(t, u) = \langle a, f\rangle$

Theorem 3.1: For any two queries q_1, q_2, if there is a unique SYP on $\mathrm{terms}(q_2)$ that yields $\mathrm{terms}(q_1)$, then $q_1 \sqsupseteq_q q_2$ iff $\mathrm{gap}(q_1) \geq \mathrm{gap}(q_2) - |\mathrm{oo}(q_1, q_2)|$

Proof We define queries $q_1 = \langle(t_1, ..., t_w), g_1\rangle$ and $q_2 = \langle(u_1, ..., u_q), g_2\rangle$. Assume that $q_1 \sqsupseteq_q q_2$. We construct a document $\hat{d}$ whose text is the terms $u_1, ..., u_q$ in order and separated by a space. It follows that $\hat{d} \in \mathrm{hits}(q_2)$. If there is no subsequence of $\mathrm{terms}(q_2)$ that contains all the terms of $\mathrm{terms}(q_1)$ in order, then document $\hat{d}$ is not hit by query q_1 which contradicts the assumption that $q_1 \sqsupseteq_q q_2$. It follows that $\mathrm{terms}(q_1) \supseteq_t \mathrm{terms}(q_2)$.

Assume that there is only one critical progression on $\mathrm{terms}(q_2)$ yielding $\mathrm{terms}(q_1)$, which we deonote $\mathrm{m} : \{1, ..., w\} \to \{1, ..., q\}$. For document $\hat{d}$ the following inequality holds:

$$\begin{aligned} q - 1 \leq g_2 \Rightarrow \\ \Rightarrow q - \mathrm{m}(w) + \mathrm{m}(w) - \mathrm{m}(1) + \mathrm{m}(1) - 1 \leq g_2 \Rightarrow \\ \Rightarrow |\mathrm{rom}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| + \mathrm{m}(w) - \mathrm{m}(1) + \\ + |\mathrm{lom}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| \leq g_2 \Rightarrow \\ \Rightarrow |\mathrm{oo}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| + \mathrm{m}(w) - \mathrm{m}(1) \leq g_2 \Rightarrow \\ \Rightarrow \mathrm{m}(w) - \mathrm{m}(1) \leq g_2 - |\mathrm{oo}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| \end{aligned}$$

Assume that $g_2 - |\mathrm{oo}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| > g_1$. We create document $\hat{d}'$ from document $\hat{d}$ by adding $g_2 - q + 1$ tokens of a newly introduced term that is neither in $\mathrm{terms}(q_1)$ nor $\mathrm{terms}(q_2)$, following the token in position $mp(1)$ in document $\hat{d}$. Document $\hat{d}'$ has exactly g_2 tokens. By adding $g_2 - q + 1$ to the left part of all above inequalities we obtain equations since this is the case where the span of the subsequence is precisely equal to the gap of the query. Document $\hat{d}'$ contains a subsequence corresponding to $\mathrm{terms}(q_2)$ and is thus hit by query q_2. The subsequence of $\mathrm{terms}(q_1)$ in $\hat{d}'$ is still unique since we introduced a non-previously existing term and its span in $\hat{d}'$ is $\mathrm{m}(w) - \mathrm{m}(1) + g_2 - q + 1 = g_2 - |\mathrm{oo}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))|$, which is in turn larger than g_1 according to the assumption. This means that document $\hat{d}'$ is not hit by query q_1 which contradicts the fact that $q_1 \sqsupseteq_q q_2$. Thus, it must be that $g_1 \geq g_2 - |\mathrm{oo}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))|$

Inversely, assume that $g_1 \geq g_2 - |\mathrm{oo}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))|$. If a document d is hit by query q_2, there is a text segment t in d for which there is a SYP $p_1 : \{1, ..., q\} \to \{1, ..., n\}$ that corresponds to $\mathrm{terms}(q_2)$. Text segment ts contains one progression that corresponds to the m-progression on $\mathrm{terms}(q_2)$ yielding $\mathrm{terms}(q_1)$, and therefore its image is a subset of the image of p_1. We denote this progression $p_2 : \{1, ...w\} \to \{1, ..., n\}$. For text segment ts:

$$\begin{aligned} p_1(q) - p_1(1) \leq g_2 \Rightarrow \\ \Rightarrow p_1(q) - p_2(w) + p_2(w) - p_2(1) + p_2(1) - p_1(1) \leq g_2 \end{aligned}$$

The gap $p_1(q) - p_2(w)$ is at least as big as the length of the right outer omission of $\mathrm{terms}(q_1)$and $\mathrm{terms}(q_1)$, since it contains the terms of the right outer omission and possibly terms of document d that are not in the outer omission. Likewise, $p_2(1) - p_1(1)$ is at least as long as the length of the left outer omission of $\mathrm{terms}(q_1)$ and $\mathrm{terms}(q_2)$. Formally:
$|\mathrm{rom}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| \leq p_1(q) - p_2(w)$ and
$|\mathrm{lom}(\mathrm{terms}(q_i), \mathrm{terms}(q_j))| \leq p_2(1) - p_1(1)$
From the three inequalities above, it follows that:

$$\begin{aligned} |\mathrm{lom}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| + p_2(w) - p_2(1) + \\ + |\mathrm{rom}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| \leq g_2 \Rightarrow \\ \Rightarrow |\mathrm{om}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| + p_2(w) - p_2(1) \leq g_2 \Rightarrow \\ \Rightarrow p_2(w) - p_2(1) \leq g_2 - |\mathrm{om}(\mathrm{terms}(q_1), \mathrm{terms}(q_2))| \end{aligned}$$

From the inequality hypothesis, it follows that:
$p_2(w) - p_2(1) \leq g_1$. Thus, there is a SYP of span less than or equal to g_1 in ts that corresponds to $\mathrm{terms}(q_1)$, hence, query q_1 also hits document d. Since this is true for every document d that is hit by query q_2 it follows that: $q_1 \sqsupseteq_q q_2$

Note that in proving the inverse direction of the theorem, we did not require uniqueness of the SYP on $\mathrm{terms}(q_2)$ that yields $\mathrm{terms}(q_1)$. This means that if there is term subsumption and the gap inequality holds we can always infer query subsumption without further checks.

B. Implementation of Query Subsumption Checking

As we explain in sections IV-C and IV-D, efficiently checking for query subsumption is important for the speedy execution of QuIET. Given a query q_i and an existing query set Q, we need a method that efficiently identifies whether there exists a $q_j \in Q$ such that either $q_i \sqsubseteq q_j$ or $q_i \sqsupseteq q_j$. We can identify query subsumption by first finding queries in Q whose terms either subsume or are subsumed by q_i, and then checking if the gap inequality of Theorem 3.1 holds. By ignoring the uniqueness assumption, we would fail to identify subsumption and keep redundant queries in the query set for some cases of queries. For example, for queries $q_1 = \langle\mathrm{enjoy}, \mathrm{coffee}, 1\rangle$, and $q_2 = \langle\mathrm{enjoy}, \mathrm{coffee}, \mathrm{enjoy}, \mathrm{coffee}, 4\rangle$ the gap inequelity fails, yet, there is query subsumption. We can use the methods proposed in [8] to check for uniqueness of the subsequence involved in the term subsumption using suffix tree indices, but this would unnecessarily inflate the implementation complexity. This is because we generate queries based on tokens that fall in the same sentence which makes such queries unlikely to appear in practice. Thus, ignoring them causes a negligible redundancy in the set of queries. Due to this, we decide term subsumption by only checking for the existence of a given subsequence in a larger string. The size of the outer omission is obtained easily by counting the tokens left and right of the subsequence identified by the term subsumption checking algorithm. Thereafter, we check the inequality of Theorem 3.1 to decide query subsumption.

To avoid checking all quadratic pairs of queries in a query set Q for query subsumption we construct set structures that allow us to check only queries that are relevant.

For every term t in the corpus we define $\mathrm{queries}(t)$ to be the set of all queries in Q that contain term t.

Term-subsuming queries (tsg) of q_i are queries in Q whose terms subsume the terms of q_i. We identify the term-subsuming queries of q_i as follows:

$$\mathrm{tsg}(q_i) := \{q_j \mid q_j \in \bigcap_{k=1}^{n} \mathrm{queries}(t_k^n) \wedge \mathrm{terms}(q_i) \subseteq_t \mathrm{terms}(q_j)\}$$

Term-subsumed queries (tsd) of q_i are queries in Q whose terms are subsumed by the terms of q_i. We identify the term-subsumed queries of q_i as follows:

$$\mathrm{tsd}(q_i) := \{q_j \mid q_j \in \bigcup_{k=1}^{n} \mathrm{queries}(t_k^n) \wedge \mathrm{terms}(q_j) \subseteq_t \mathrm{terms}(q_i)\}$$

Thus, if a query q_j fails to satisfy the set membership conditions of the above conjunctions we can infer non-subsumption without invoking the term subsumption checking algorithm.

C. Categorization Measures

Precision, *recall* and the *F-measure* are used as measures of the performance of a classification method. A classifier may erroneously classify a non-member of target category c as positive and a member of the category c as negative. These documents are called false positives and false negatives respectively and define two sets that we denote FP and FN. The correctly classified positive documents are the true positives and form set TP and the correctly classified negative documents are the true negatives and form set TN. Using these sets we get the following definitions: $\mathrm{precision} = \frac{|TP|}{|TP|+|FP|}$, that is, precision is the proportion of the truly positive documents in the set of documents that the classifier deems positive, and $\mathrm{recall} = \frac{|TP|}{|TP|+|FN|}$, that is, recall is the fraction of the number of true positive documents that the classifier returns over the number of the actual documents of category c in the corpus.

Precision, denoted as p, and recall, denoted as r, are not good measures of classification performance alone. For example, we can obtain a recall of 100% by trivially labeling all documents in the corpus as positive. F_1 is the harmonic mean of precision and recall: $F_1 = 2 \cdot \frac{p \cdot r}{p+r}$. It is a special case of $F_\beta = (1+\beta^2) \cdot \frac{p \cdot r}{\beta^2 \cdot p + r}$, where β is a non-negative real. In our study, in addition to F_1, we use $F_{0.5}$ which places an emphasis on precision.

IV. The QuIET algorithm

The input of the training phase of QuIET is a set of labeled texts which are used as the training set. The labels denote membership or non-membership in a target category c. The output of the training phase is a set of queries and associated scores. These query sets are then used to categorize unlabeled texts of the same nature. By building one such model for every target category, we obtain a tool that can tag unknown texts as belonging to multiple categories. Below we describe the method for a single target class c. To build classifiers for additional classes, we repeat the same process for a different class. The process involves four steps: feature selection, query generation, query selection, and categorization of unlabeled texts.

Initially, the training set of documents undergoes the standard normalization and stemming of words to ascertain that different forms of the same word are not counted as occurrences of different words. Thereafter, the method goes through the texts and identifies the words that are the most informative for binary classification according to a separation metric. The selected words form the set of features.

We use these features to generate queries by going over all sentences of the documents in the training set. We calculate the number of documents that each query hits and select only the queries that return documents that are members of category c with high precision. Once a set of high-precision queries has been generated, we sort the queries by their recall in descending order, and start selecting queries that are not already subsumed by some other query in the final set. If we identify a query that subsumes other existing queries in the final set, we replace the subsumed query with the subsuming query.

Finally, for the final set of selected high-precision and most general queries we calculate a normalized score based on a separation metric. The model is the final set of queries and their scores. This allows us to calculate a threshold query score, denoted as θ, that separates the training set optimally according to a classification measure (see Section III-C for measures of classification).

To classify an unseen document we calculate a normalized document score based on the score of the queries of the model that hit the document. QuIET considers the documents whose document score exceeds threshold θ as members of category c and the rest as non-members of the category. Below we give a more detailed explanation of the method's steps.

A. Feature Selection

Feature selection, a standard step in text categorization, allows us to only consider words that tend to be indicative of the target class in query generation. For each word in the corpus of labeled documents, we calculate a score corresponding to the significance of the word in the category of interest and select the words with the highest scores as features.

There are several scoring functions that we can use to compute the score of the words in a training corpus. Prominent examples are the Chi-Squared statistics test, Information Gain, Document frequency, Odds Ratio and Log Probability Ratio. In [9] and [10], extensive comparative results that study the performance of the above methods are reported. In [11], Foreman introduced a new metric called Binormal Separation. The metric outperformed other metrics in most cases, while its lead was wider for cases of elevated category skew, which is rampant in text categorization. For this reason, we use Binormal Separation (BNS) as the separation metric for feature selection. We calculate the BNS score of each word in the corpus, sort the words based on the score, and pick the desired number of top words to form feature set F.

B. Query Generation

After the feature selection, we iterate over the documents that belong to the target category and generate multi-term queries comprising the features extracted in the previous stage.

Given a piece of text containing n words, we can generate up to $\binom{n}{r}$ multi-term queries containing r terms. As the number of words increases there is a combinatorial explosion of generated queries. To address this, we limit r to be less than 4. We also split the text into sentences and generate queries using the features that are present in each sentence. We put a hard cap on the number of words that a sentence can have. If a sentence exceeds 15 words we artificially split it in two. We generate all the combinations of one-, two- and three-term queries in every sentence, therefore the maximum number of queries per sentence is limited to a constant. Thus, the total number of queries is asymptotically linear to the size of the text corpus in the worst case.

C. Query Selection

To create a set of selected queries for binary classification, we initially iterate over all positive documents in the training corpus, that is, all documents that are labeled as belonging to class c. For each document, we use the method we described above to obtain all combinations of queries with terms belonging to the feature set. We evaluate the queries for each document in the document index to calculate the precision of each query. Precision of a query is the number of documents belonging to target category c in the training corpus that the query hits as a percentage of the entire set of documents of the training corpus that it hits. We only retain documents that pass a minimum precision threshold. We generally choose the threshold to be high, e.g. 90%. We can tune the method for various precision thresholds to select the one that provides the best final results. We then calculate a score for each of the high-precision queries. The score indicates the level of separative effectiveness of the query. A good option for the score function is the same binormal separation score (BNS) mentioned in Section IV-A, which we use in our implementation.

We check generated queries for subsumption so that the most general queries are selected and less general queries are ignored. In this way, we increase the generalizability of the final query set. Figure 1 shows the pseudocode for the process of query generation and selection.

```
Function select_queries
Input  : feature_set: a set of unigrams, document_index : an
         index of documents, precision_threshold : a real
         number, c: a document category
Output: A set of scored queries
Data: queries: a set of queries, scored_query_set: a set of
      scored queries, precision, recall, score: real numbers
scored_query_set ← ∅;
foreach document d of class c in document_index do
    foreach sentence s in d do
        queries ← generate_queries(feature_set, s);
        foreach query q in queries do
            precision ← calculate_precision(q,
            document_index, c) ;
            if precision ≥ precision_threshold then
                score ← calculate_score(q);
                recall ← calculate_recall(q);
                scored_query_set.add(q, recall, score);
sorted_queries ← sort_by_recall_descending(scored_query_set);
selected_query_set ← sorted_queries[0];
foreach i in [1..size(sorted_queries)] do
    q ← sorted_queries[i];
    if subsuming_queries(q, selected_query_set) != ∅ then
        continue;
    Q ← subsumed_queries(q, selected_query_set);
    if |Q| > 0 then
        remove_all(Q, selected_query_set);
    selected_query_set.add(q, selected_query_set);
return selected_query_set;
```

Fig. 1: Algorithm for query selection

D. Categorization of Unlabeled Texts

After computing the scores of all selected queries we calculate a score for each document in the training corpus. For each document, we initially obtain a raw score as the sum of the scores of the queries in the selected query set Q that hit the document: $raw\_score(d) = \sum_{q \in Q \text{ s.t. } d \in \text{hits}(q)} q.score$

The distribution of the raw scores usually contains outliers. If we normalize using the range of raw scores the majority of documents cluster close to zero. We use normalization factor: $z = Q(3, \text{raw\_scores}) + 1.5 \cdot IQR(\text{raw\_scores})$, where $Q(3, \text{raw\_scores})$ returns the value at the third quartile of the document raw scores seen in the training data, and method IQR returns the inter-quartile range of the raw scores. This way, we are effectively setting our normalization factor z to the maximum non-outlier raw score that exists in the training set of documents. This allows us to compute a normalized score using the formula: $\text{score}(d) = \min(\text{raw\_score}(d)/z, 1)$. We sort the documents of the training set based on their scores. Documents with higher scores are more likely to be positive documents, while documents with lower scores are likely to be non-positive. We calculate a threshold θ of the document scores that separates the documents optimally according to a classification metric. For example, if we use the F- 1 measure, the set of documents that has score that is equal or larger to θ has precision and recall that corresponds to the highest F-1 measure compared to that of any other choice of score threshold.

We use the selected set of scored queries Q and the threshold θ to label unlabeled texts as members or non-members of target class c. For each unseen document we calculate its raw score according to the formula above. We then normalize using the normalization factor z that we have obtained from the training set. If the document's normalized score is equal or larger to the threshold θ, we label it as belonging to target class c, otherwise we label it as a non-member of c. Running all the queries of Q on each document during classification can

be computationally expensive because typically the number of queries that hit a document is far less than the number of queries in the query set. During classification, our implementation generates queries from the document being classified and checks those queries for subsumption by the queries in the query set. This way, we can determine which queries of the QuIET query set hit the document without actually running every query on each document. Figure 2 shows the pseudocode of the classification algorithm.

```
Function classify
Input  : model: a set of query-score pairs, Feature_Set: a set of
         unigrams, score_threshold: a real number, z: a real
         number, document : a document
Data: score: the running score
Output: classification
score ← 0;
foreach sentence s in document do
    foreach query q in generate_queries(Feature_Set, s) do
        foreach q' in subsuming_queries(q,model) do
            score ← score + q'.score;
score ← min(score/z, 1);
if score ≥ score_threshold then
    classification ← 1;
else
    classification ← 0;
return classification;
```

Fig. 2: The classification algorithm

E. Active Learning with QuIET

Active learning is a technique widely used in machine learning and artificial intelligence [12]. In a naive setting, a training algorithm randomly picks data points from a wider pool and assigns them to human annotators. The annotated data are then used to train the model, and the model can then in turn be used to classify non-annotated data. This may be wasteful since some of the data we choose to annotate may be less informative than others. Active learning is an umbrella term for several techniques that perform the training in iterations and augment the training set by smartly picking instances from the pool that are likely to be more informative.

A simple active learning approach, applicable to techniques such as linear regression and Support Vector machines, is to pick training instances in proximity to the current decision boundary. Active learning can lead to significantly fewer number of manual annotations and lower the cost of building the model. Previous studies have explored active learning for text categorization purposes [13] [14] and data of highly imbalanced categories [15] with the use of Support Vector Machines.

Inspired by the use of active learning in machine learning techniques, we implement an active learning approach for QuIET. We view threshold θ as a decision boundary in a one-dimensional space. To apply active learning to QuIET, we perform the query generation and selection in iterations. We pick a small set of documents whose score is close to the current threshold θ and add them to the training corpus in every iteration.

V. Experimental Study

A. Methodology

We use two datasets to evaluate the performance of QuIET. One is the Modapte split of the Reuters-21578 text dataset, first used in [2], and subsequently in several studies. Our goal is to evaluate QuIET on a known curated dataset. We compare the results with SVM classifiers using RBF kernel function, with optimally tuned C and γ for every category.

Subsequently, we present the results of QuIET on noisy local businesses website data for a Groupon-specific set of categories describing local businesses, contrasted with results on the same data using SVM classifiers.

We thereafter evaluate the efficiency of the active learning variant of QuIET. We implement two different approaches for active learning with QuIET, one eager and one lazy.

The eager approach builds the model using an initial random set of 50 documents, then picks an additional 50 documents and rebuilds everything from scratch, including the feature set. It builds the query set using the new feature set and the entire current training corpus. It selects the queries by evaluating every query on every document in the corpus. It continues to augment the training set by 50 documents until all the documents in the pool are included in the training set.

The lazy approach builds everything from scratch only periodically and not in every iteration. In most iterations, it maintains the previous feature set and keeps the set of older queries and their associated scores. It augments the query set with new queries which are generated only using the documents that are added to the training set in the current active learning step. The reasoning behind this approach is that for few additional items in the training set, we expect small changes in the feature set and only few existing queries that will fall below the precision threshold. We thus evaluate fewer queries on the corpus and optimistically assume that existing queries still qualify. This approach can lead to slower increase in performance between active learning iterations but it can save significant time, since query evaluation is the major bottleneck in the construction time of the QuIET models. In this implementation, we set the ratio of lazy to eager steps to be 9:1, that is, we retrain everything from scratch once in every 10 active learning iterations.

We ran QuIET for a range of values of the precision threshold for query selection to determine the optimal value. We obtain the best results for both F_1 and $F_{0.5}$ in the vicinity of precision threshold 0.90 for most categories and datasets. We can obtain the best results by tuning individually for each dataset which can lead to different optimal precision thresholds. However, in a real setting, we do not have a labeled test set, and we cannot know the optimal threshold unless the test set is labeled. Thus, in real applications, we must use a fixed precision threshold that is likely to produce good results based on previous test results.

B. Reuters Dataset

In Table I we can see the comparative performance of SVM with RBF kernel and QuIET for the most populous categories of the Modapte split of the Reuters dataset. QuIET uses a precision threshold of 90% for query selection.

Method	Category	Precision	Recall	F_1	$F_{0.5}$
SVM	earn	0.98	0.99	*0.98*	*0.98*
QuIET	earn	0.982	0.912	*0.95*	*0.97*
SVM	acq	0.95	0.94	*0.94*	*0.95*
QuIET	acq	0.929	0.792	*0.85*	*0.90*
SVM	crude	0.79	1.0	*0.88*	*0.83*
QuIET	crude	0.871	0.752	*0.81*	0.84
SVM	corn	0.8	1.0	*0.89*	*0.84*
QuIET	corn	0.901	0.72	*0.80*	*0.86*

TABLE I: Comparative performance for Reuters dataset

We observe that QuIET has a comparable performance against Support Vector Machines. We obtained similar results for other categories of the Reuters dataset. The results of the SVM classifiers are the best among a variety of tuning parameters of the kernel function for each category and test set, which is not applicable in a real setting as mentioned in the methodology section above. On the other hand, we report QuIET results for a fixed precision threshold parameter of 90%. If we have the flexibility to alter the precision threshold parameter to the optimal for each category, we achieve a boost in QuIET performance for several categories and test sets. We use RBF kernels, since they produce results that are similar to the best reported in the literature. Results for the Reuters dataset using different kernels, such as String Kernels [16] are comparable to SVM classifiers with RBF, having slightly superior performance on some categories and slightly inferior on others, but not demonstrating a generally superior performance.

QuIET maintains the advantage of human comprehensibility of the query sets that it produces. The results suggest that QuIET is an efficient method for text categorization that can be used either as a stand-alone text classifier or as an extra vote to boost the performance of ensemble classifiers [17].

C. Local Businesses Website Data

Groupon merchant taxonomy comprises a hierarchy of merchant categories. Categories lower in the hierarchy which we call 'services' are often more fine-grained and among available data there is generally more diversity and noise. We obtain the categories of each local business through the use of crowdsourcing. The low quality of the training set is due to both the inherent noise that is ubiquitous in online content but also due to the existence of errors in the crowdsourced labels. For this kind of datasets, machine learning text classification methods perform poorly. As an example, a merchant under the high-level category 'Health & Beauty' may or may not provide fish pedicure services. The fish pedicure service is thus under the general category 'Health & Beauty' in the taxonomy. Traditional machine learning methods achieve high performance for the high-level category 'Health & Beauty' but they give poor results for the 'fish pedicure' service. For four such service categories that inhabit lower floors in the taxonomy, Table II shows the results of the comparative performance of QuIET and SVM with radial basis kernel function tuned for optimal performance. The size of the feature set is $40,000$.

Method	Category	Precision	Recall	F_1	$F_{0.5}$
SVM	A	0.231	0.0536	*0.09*	*0.14*
QuIET	A	0.417	0.179	*0.25*	*0.33*
SVM	B	0.371	0.425	*0.40*	*0.38*
QuIET	B	0.42	0.38	*0.40*	*0.41*
SVM	C	0.24	0.19	*0.21*	*0.23*
QuIET	C	0.36	0.21	*0.27*	*0.32*
SVM	D	0.609	0.955	*0.74*	*0.66*
QuIET	D	0.755	0.627	*0.69*	*0.73*

TABLE II: Comparative performance for local businesses website data

The performance is low for both methods, yet, we observe that QuIET outperforms SVM for all four categories. The results suggest that for heavily noisy data, a common problem for data obtained online, a query-based text categorization algorithm such as QuIET is likely to produce better results than traditional machine learning techniques in some cases. Also, QuIET places an emphasis on precision (due to the high-precision queries which comprise the QuIET model), which is often a desideratum.

Threshold θ which acts as a decision boundary can be manually fine-tuned by the users of QuIET. This is because QuIET calculates θ using the training set and may not always generalize optimally for a given test set. For example, for Category D, while the training set based threshold θ achieves a 69% F-1 measure performance on the test set, the optimal separation on the test set of documents sorted by their score reaches 84%.

D. Performance of Active Learning with QuIET

For Category D, we have $1,650$ documents in the training set. In Figure 3 we demonstrate the performance of the active learning variant of QuIET for Category D. The results we obtained for other categories are qualitatively similar. We report the F-1 measure for three different methods of training QuIET in iterations. For all three methods, the training starts with a random subset of the training set containing 50 documents. Due to this random initialization, we report the average result from 3 executions of each experiment.

The non-active method picks an additional 50 documents at each iterative step randomly from the training set and retrains. As the size of the training set increases, we observe that the non-active QuIET gradually increases performance until it reaches the final performance of 69% (same as reported in Table II).

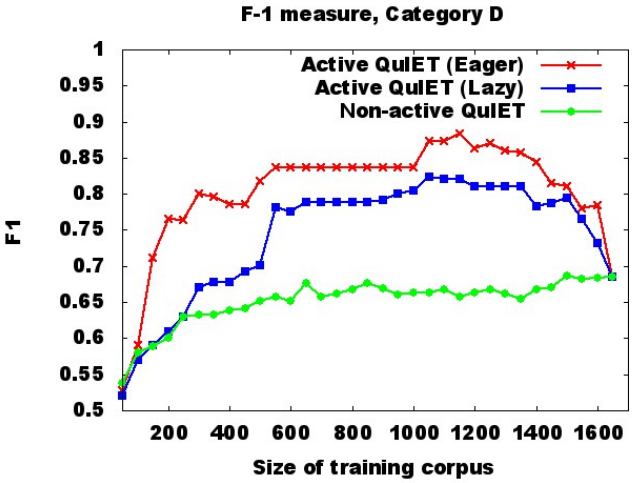

Fig. 3: Performance of QuIET with active learning

Active QuIET surpasses the performance of non-active QuIET by using a fraction of the training set. The performance of the eager variant gradually increases, it achieves a maximum performance that approaches 88% using roughly two thirds of the size of the training corpus, and starts gradually decreasing until it reaches the final performance of 69%. The lazy variant demonstrates a similar behavior, outperforming non-active QuIET but achieving a lower maximum than that of the eager implementation.

The reason behind this boom and bust is the presence of spam and wrongly classified examples in the training set that are isolated in the active learning process. For example, the training set may contain a spam webpage that is correctly labeled as a non-member of the category, yet, the content may contain text segments that are common in positive examples of the category and are therefore hit by the same query. Given that the content of the website is bogus, it is unlikely that its score value is near the current score threshold θ, and will therefore not be picked during the active training process. When the spam webpage is eventually included in the training set it may cause some of the existing queries that hit it to fall below the precision threshold and be excluded from the query set despite the fact that these queries may be useful in separating categories. This way, the generality and quality of the QuIET model decreases.

The lazy method achieves a lower boost in categorization quality because of failing to properly update the query set in some cases, but the training occurs much faster. In particular, the decrease in construction time can be in the order of 9x (the ratio of lazy to eager steps). The construction of a QuIET query set involves the evaluation of thousands of queries on each document of the training set and typically requires several hours of execution time using a modern processor. The eager active learning repeats the construction from scratch at each iterative step and may require several days to complete. The speedup of the lazy variant is therefore significant. Lazy active learning can thus be the method of choice for quickly training numerous binary classifiers for a taxonomy involving many categories, even if the derived query sets achieve slightly reduced categorization performance compared to the eager implementation.

VI. Conclusions

QuIET is a new method that produces human understandable text classifiers. Training of QuIET scales to large datasets while its categorization efficiency outperforms machine learning techniques for some noisy datasets. The method is suitable for large sets of documents obtained from online sources where noise and spam is ubiquitous. Active learning is applicable to QuIET and can boost its performance in many cases.

References

[1] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, Mar. 2002.

[2] C. Apte, F. Damerau, and S. M. Weiss, "Towards language independent automated learning of text categorization models," in *Proceedings of the 17th annual ACM/SIGIR conference*, 1994, pp. 23–30.

[3] E. Frank and R. R. Bouckaert, "Naive bayes for text classification with unbalanced classes," in *Proceedings of the 10th European Conference on Principle and Practice of Knowledge Discovery in Databases*, ser. PKDD'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 503–510.

[4] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, ser. ECML '98. London, UK, UK: Springer-Verlag, 1998, pp. 137–142.

[5] S. Bloehdorn and A. Hotho, "Boosting for text classification with semantic features," in *Proceedings of the MSW 2004 Workshop at the 10th ACM SIGKDD conference of knowledge discovery and data mining*, 2004, pp. 70–87.

[6] N. Tripathi, M. P. Oakes, and S. Wermter, "Hybrid classifiers based on semantic data subspaces for two-level text categorization," *Int. J. Hybrid Intell. Syst.*, vol. 10, no. 1, pp. 33–41, 2013.

[7] L. Hirsch, R. Hirsch, and M. Saeedi, "Evolving lucene search queries for text classification," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '07. New York, NY, USA: ACM, 2007, pp. 1604–1611.

[8] J. Pei, W. C.-H. Wu, and M.-Y. Yeh, "On shortest unique substring queries," in *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ser. ICDE '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 937–948.

[9] D. Mladenic and M. Grobelnik, "Feature selection for unbalanced class distribution and naive bayes," in *In Proceedings of the 16th International Conference on Machine Learning (ICML.* Morgan Kaufmann Publishers, 1999, pp. 258–267.

[10] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization." Morgan Kaufmann Publishers, 1997, pp. 412–420.

[11] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1289–1305, Mar. 2003.

[12] B. Settles, "Active learning literature survey," *University of Wisconsin, Madison*, 2010.

[13] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *J. Mach. Learn. Res.*, vol. 2, pp. 45–66, Mar. 2002.

[14] B. Yang, J.-T. Sun, T. Wang, and Z. Chen, "Effective multi-label active learning for text classification," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: ACM, 2009, pp. 917–926.

[15] M. Bloodgood and K. Vijay-Shanker, "Taking into account the differences between actively and passively acquired data: The case of active learning with support vector machines for imbalanced datasets," ser. NAACL-Short '09, 2009, pp. 137–140.

[16] H. Lodhi, J. Shawe-taylor, and N. Cristianini, "Text classification using string kernels," *Journal of Machine Learning Research*, vol. 2, pp. 563–569, 2002.

[17] L. Rokach, "Ensemble-based classifiers," *Artif. Intell. Rev.*, vol. 33, no. 1-2, pp. 1–39, Feb. 2010.