USING MACHINE LEARNING TECHNIQUES TO LABEL LABORATORY TEST RESULTS (PHASE II)

Tae Yoon (Harry) Lee
taeyoon.lee@alumni.ubc.ca

Yu (Iris) Gao
gaoyu226@gmail.com

Jackie Lam
jackielam918@gmail.com

January 5, 2020

Contents

1 Introduction

This report summarizes the collaborative work between the 2019 UBC Data Science for Social Good (DSSG) fellows and the British Columbia Centre for Disease Control (BCCDC).

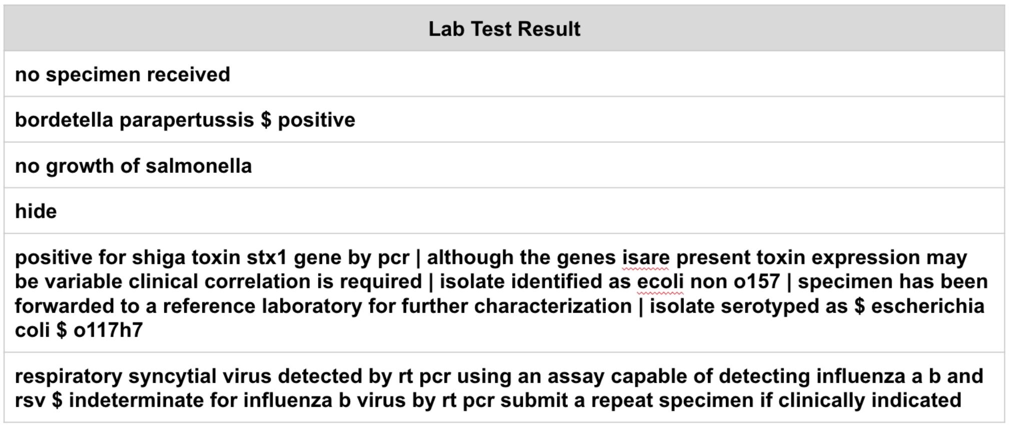
1.1 The BCCDC

One of the responsibilities of the BCCDC is to review and extract information from laboratory test results. The extracted information is used to identify cases that meet criteria for reportable communicable diseases for disease surveillance and for population health research purposes. The challenge lies in extracting information efficiently due to the nature of the laboratory test result data. The lab test results are written by lab technicians, who can use a code that generates the text automatically (the code is registered by the lab technician), or input free-form text, or input a combination of those two. Consequently, the lab test results are semi-structured text, and they inevitably contain human-made errors. More importantly, no simple rule-based method can be used on the laboratory test results to extract information. Currently, the lab results are being labeled more or less manually by domain experts. In particular, for each lab result, three targets of interest need to be extracted:

- test_performed: an indicator whether a test for an organism of interest is performed.
- test_outcome: an outcome of a test. Its levels are positive, negative, missing, and indeterminate.
- organism_name: the name of an organism that is being tested for.

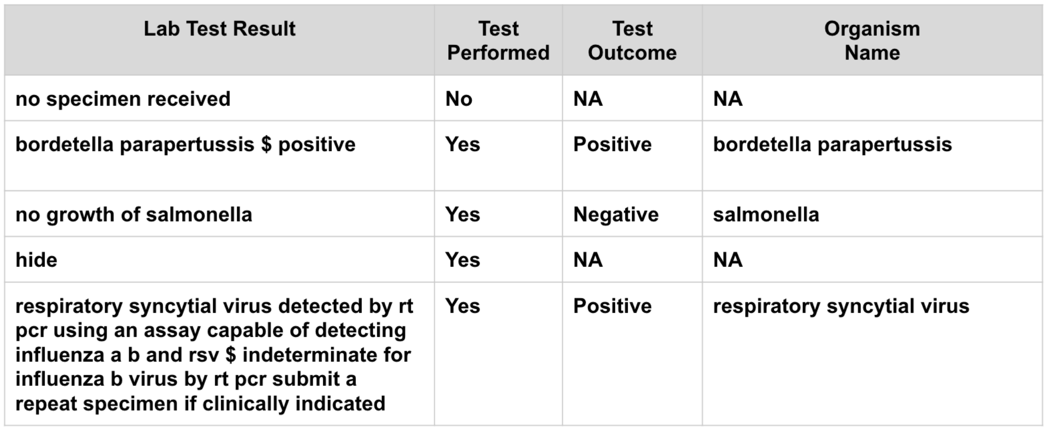
Figure 2 shows the examples of the labeled lab test results from Figure 1.

The manual labeling process is too expensive and time-consuming for the BCCDC. While a simple solution is to enforce a rule to write lab test results in a structured way (due to technical difficulties, this is not feasible at the moment), the historical lab test results need to be labeled anyway. In search for an automated process to label the lab test results, the BCCDC and the 2018 DSSG fellows conducted a preliminary investigation of applying machine learning methods (Chen *et al.*, 2018). The 2018 DSSG fellows built a pipeline for analyzing the laboratory results using machine learning classifiers and showed that the machine learning classifiers achieve a high level of performance in terms of (unweighted) accuracy for test_performed (99%) and test_outcome (98%).



Lab Test Result
no specimen received
bordetella parapertussis $ positive
no growth of salmonella
hide
positive for shiga toxin stx1 gene by pcr | although the genes isare present toxin expression may be variable clinical correlation is required | isolate identified as ecoli non o157 | specimen has been forwarded to a reference laboratory for further characterization | isolate serotyped as $ escherichia coli $ o117h7
respiratory syncytial virus detected by rt pcr using an assay capable of detecting influenza a b and rsv $ indeterminate for influenza b virus by rt pcr submit a repeat specimen if clinically indicated

Figure 1: Examples of laboratory test results.

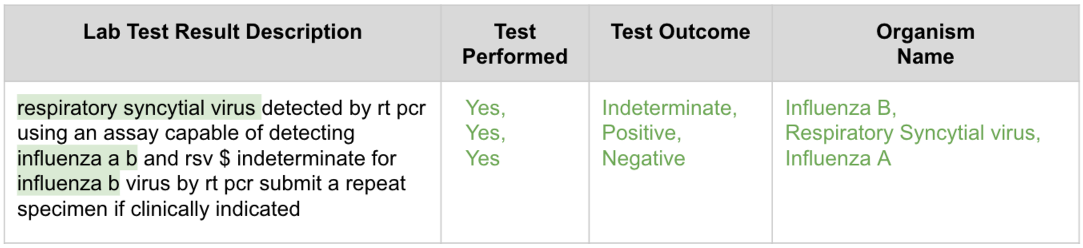


Lab Test Result	Test Performed	Test Outcome	Organism Name
no specimen received	No	NA	NA
bordetella parapertussis $ positive	Yes	Positive	bordetella parapertussis
no growth of salmonella	Yes	Negative	salmonella
hide	Yes	NA	NA
respiratory syncytial virus detected by rt pcr using an assay capable of detecting influenza a b and rsv $ indeterminate for influenza b virus by rt pcr submit a repeat specimen if clinically indicated	Yes	Positive	respiratory syncytial virus

Figure 2: Examples of labeled laboratory test results.

1.2 Contributions

Building upon the last year's work, our initial goal, as suggested by the BCCDC, was to improve the existing pipeline in terms of accuracy and computational efficiency and to assess the use of a terminology dictionary for the task of labeling organism_name. We started with learning about the data at hand. We delved deeply into the data and postulated potentially useful features based on summary statistics and visualisations (Section 2). We then examined and critically evaluated the existing pipeline. In addition to incorporating the additional features obtained by our exploration of the data, we made several improvements on the pipeline. The most important improvement is implementing hyperparameter optimization (Section 3.10); results of using the additional features and hyperparameter optimization are given in Section 4.1. At the same time, we realized that the machine learning methods used in the last year are not appropriate to help the BCCDC meet the ultimate goal of labeling lab test results when there exists more than one organism of interest in a lab test result. The machine learning methods in the existing pipeline do not have the capability to identify multiple organism names and their corresponding outcomes in a lab test result, such as the last row of the table in Figure 2; see Figure 3 for the fully labeled version of it. In Section 3.8, we propose an appropriate machine learning method based on neural networks to solve that problem. However, we encountered another common problem in machine learning. The BCCDC did not have a fully labeled (training) dataset of lab test results containing more than one organism of interest. In Section 3.9, we introduce a framework, called active learning, for sampling unlabeled data efficiently to create a training dataset. Another advantage of this framework is that the proposed machine learning model is trained at the same time. The active learning framework would allow the BCCDC to label a smaller amount of data to achieve the same level of performance. As this new objective of fully labeling lab test results with multiple organisms was defined towards the end of the project, we built a business case of using the active learning framework with the proposed machine learning method on a simple labeled dataset that only contains one organism of interest, Hepatitis C virus (see Section 2.5 for the description of the dataset). In Sections 4.2 and 4.3, we illustrate that the proposed machine



Lab Test Result Description	Test Performed	Test Outcome	Organism Name
respiratory syncytial virus detected by rt pcr using an assay capable of detecting influenza a b and rsv $ indeterminate for influenza b virus by rt pcr submit a repeat specimen if clinically indicated	Yes, Yes, Yes	Indeterminate, Positive, Negative	Influenza B, Respiratory Syncytial virus, Influenza A

Figure 3: An example of fully labeled laboratory test results.

learning method actually performs better than the machine learning methods used in the existing pipeline and that a much smaller sample size is required to achieve a high level of performance with the active learning framework.

1.3 Data Science for Social Good

The Data Science for Social Good (DSSG) Program at the University of British Columbia (UBC) is a 14-week summer fellowship program hosted by hosted by the UBC Data Science Institute and sponsored by Boeing, CIFAR, and MITAS. It is designed to provide an interdisciplinary data science research experience in collaboration with an organisation (BCCDC for this project) for undergraduate and graduate UBC students across different disciplines with the aim of bringing benefits to the society. At the end of the report, we discuss how our work would bring social good and how we envision the proposed framework would be implemented.

2 Data

The data was extracted from electrical clinical records of individual tests conducted by the BCCDC's lab, which was mainly unstructured free form text data. Sensitive personal information was removed due to privacy protection protocol.

The original dataset includes records from multiple lab tests each aiming to identify certain type(s) of microorganism. The initial goal of this project is to automate the classification process for the three targets described in Section 1.1, with the targets $test\_outcome$ and $organism\_name$ dedicated to the most critical microorganism tested; as the project proceed, we naturally come to a new objective that requires finding the corresponding test outcome for multiple microorganisms involved in a lab test, which can not be accomplished by the current machine learning method. Due to the complexity of the new objective, we apply only a subset of the original data to tailor the deep learning method, which will be described in Section 2.5.

The original dataset consists of about 400,000 test results, each composed of the text of a lab report along with other meta-data, all stored in the database columns. More specifically, the text of a lab report is stored as *result_full_description*. A *result_full_description* is made of several observations, stored as *result_description*, which is a block of text that is appended to *result_full_description* one at a time, separated by pipes (as shown in Figure 1.). A lab report has the following essential metadata:

- The *test_key* and *result_key* that are generated by the system as a hash code.
- The *test_type*, *test_sub_type*,*test_name*, *test_code* that describes a specific test routine.
- A system generated hash code, *result_tech_key*, for the lab technician who composed the test record.
- The ID of the lab record system that the test result comes from, stored as the *dss_instance_id*.

In the database, each lab report is identified uniquely by a pair of *test_key* and *result_key*, thus all data rows that have the same *test_key* and *result_key* pair have identical metadata. Therefore, concatenating all observations of the same *test_key* and *result_key* pair, ordering by the sequence number, gives us the full original lab report.

The three targets of our classification task are stored as *test_performed*, *test_outcome* and *level_1* and *level_2*. It is worth noting that the genus and species type of the tested microorganism are stored separately in *level_1* and *level_2*. If multiple microorganisms are involved in a lab test, then only the most crucial organism is reported. The *test_performed* target has two classes, yes and no. Intuitively, if *test_performed* is yes, then the lab test was carried out successfully. On the contrary, *test_performed* will be marked as no if the lab test did not happen because accidents such as broken specimen. The *test_outcome* target has four classes: positive, negative, indeterminate and missing. A positive *test_outcome* indicates the organism specified in *level_1* and *level_2* is found positive in the test. A negative *test_outcome* can be interpreted similarly. If it is unclear whether the microorganism is present or not, then *test_outcome* is marked as indeterminate. Moreover, if the *test_outcome* of a lab report is not yes, then *level_1* and *level_2* are automatically marked as *Not found*.

One challenge of this project is the serious class imbalance, as displayed in Figure 4 and Figure 5. For example, for the test outcome target, there are rarely instances for the intermediate class. As for the organism name target, the 'Not found' and 'Hepatitis C Virus' classes take up the majority of the whole dataset while all remaining classes only have a few instances. To account for such imbalance, we introduce a customized performance metric as described in Section 3.4.2.

Test Outcome	Proportion of the entire data
Positive	0.17
Negative	0.13
Missing	0.69
Indeterminate	0.01

Figure 4: Test Outcome Class Distribution

Organism Name	Proportion of the entire data
Not found	0.48
Hepatitis C Virus	0.22
Streptococcus	0.07
Other 24 classes	0.23

Figure 5: Organism Name Class Distribution

2.1 Machine Learning Feature Extraction

For the purpose of applying machine learning, we extract from the records certain fields that are considered helpful features in the classification task. Below is the set of features that are used in the classification scheme:

1. Test Type: A specific type of lab procedure or methods for conducting a clinical test. For example, one's test type can be PCR, culture and so on.
2. Lab Technician ID: It is considered as a useful feature because each lab technician has a specific writing style.
3. Number of observations: Lab technicians composed the record in multiple blocks of text as shown in Figure 1. When the lab test result is positive, the lab technicians generally compose more text than when it is negative, in order to describe more details of the finding.
4. Result Full Description: Block of text that describes the procedure and the outcome of the entire lab test.
5. Candidates: A list of viruses, bacteria or eukaryotes found in result full description annotated by MetaMap (introduced in the following section).

2.2 Bag-of-Words

In order to prepare the words in the result full description to be used in our classifiers, we need a numerical representation of the results descriptions. We will be using a Bag-of-Words approach to represent the result descriptions. The Bag-of-Words method works by count vectorizing each result description. This will create a vocabulary of words obtained from all the words in our corpus of result descriptions. This will transform our data set such that for each observations, the feature space is spanned by the vocabulary, and the corresponding values for each word will be count of the given word within the given result description. In addition, we will also be count vectorizing the results of MetaMap (Section 2.3) and append outputted candidates to our feature space. Immediately, some potential downfalls of this approach is the data set will become extremely sparse, and by only using the counts, the order of the words are lost.

2.3 MetaMap

One core difficulty of this classification problem is that the text input contains many clinical named entities and a huge variety of acronyms, especially the naming for microorganisms. As a counter method, we use MetaMap, a biomedical text annotation tool, as a helper to interpret the semantics of our clinical text data. Metamap uses knowledge-intensive approaches such as symbolic approach, natural-language-processing and computational linguistics to map text into Unified Medical Language System (UMLS) concepts. For each lab test to classify, we annotate its result full description with MetaMap and use the identified microorganisms as a machine learning feature.

2.4 Data Preprocessing

Raw data is queried from the BCCDC's SQL server in a table format. Since the result full description field is free-form text, the data needs to be processed prior to classification to remove noises. Furthermore, certain machine learning features discussed in section 2.1 are not originally contained in test records and need to be extracted from raw data. The preprocessing steps are as follows:

Preprocessing steps for the description field:

1. Deduplicate and remove empty entries.
2. For each observation in a result full description, convert all the words to lowercase and remove all characters that are not letters, numbers or spaces.
3. Replace all numeric words with token "_NUMBER_".
4. Replace all date specifying words with token "_DATE_".
5. Remove sentences that contain "previously reported as" (which indicates that the conclusion following has been overwritten and is no longer valid) so that the false report does not affect classification.
6. Concatenate observations into result full description based on sequence number. Using '$' to separate sections in an observation and '|' to separate observations.

General preprocessing steps:

1. Generate a new field "Number of Observations" for each test by counting the number of '|' in each result full description.
2. Use MetaMap to annotate each test's result full description and create a new field "Candidates" that contains the list of microorganisms found by MetaMap.

2.5 HCV data

During later stage of the project, a new objective arises to identify all microorganisms and their corresponding test outcome in a lab test. It comes to our realization that the current supervised learning scheme is not suitable for the task and deep learning would be a better approach; however, since many lab tests in the existing data involves a group of microorganisms that are not labelled and thus are not feasible for multi-class classification, the BCCDC provides us with a new data set containing lab tests for only one known microorganism: Hepatitis C Virus. The HCV data is dedicated to developing the deep learning module.

This new dataset is processed using the same steps described in section 2.4. Since the deep learning method is developed to be possibly modified into a multi-class method, we are suggested to replace the microorganism name in the description text with a token. In the HCV dataset, for example, we replace all the occurrence of the name Hepatitis C Virus along with its various acronyms with the token "TARGET" prior to training.

3 ML Techniques

In this section, we will go over the machine learning methods we will employ to solve our classification tasks. We are faced with both a binary classification problem in the case of *test_performed*, and a multi-class classification problems for *test_outcome* and *organism_name*. There are many classification algorithms that are appropriate for these classification tasks. All of the classifiers will take in as input the feature matrix created by count vectorizing the *result_full_description* of the data set in order to perform classification. However each classifier will differ in how it utilizes the inputs to produce its final prediction. We will utilize both linear and non-linear mapping of the feature matrix to the target variables. Linear models assumes that the target variable can be expressed as a linear combination between the features (the words that make up the *result_full_description*). The linear models that we will use include Support Vector Machine and Logistic Regression. As the name suggests, non-linear models performs a non-linear mapping from the feature vector to the target variable. The non-linear models we will use include Random Forest and Adaboost. Detailed explanations of each classifier used will be expressed in the next section. For each target variable, in order to select the best classifier for the task, we will perform 5-folds cross validation. At each fold, each classifier will be trained using the 80 percent of the training data, reserving 20 percent for validation. The classifier with the best average cross validation score (accuracy or f-score) will be selected. The selected classifier will then we trained on the entire training set and tested on the validation set.

3.1 Naive Bayes

The Naive Bayes classifier is a probabilistic classification model that is based on the Bayes Classifier that aims to compute the posterior $P(y|x)$ given the conditional distribution $P(x|y = y_i)$ and the prior $P(y)$ of the training data set. The posterior can be computed using Bayes Rule:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{1}$$

$$\tag{2}$$

One method to compute the conditional distribution $P(X|y = y_i)$ is to assume the features X is drawn from some distribution, for example Gaussian. In our case, our features will be the counts of each word in each *result_full_description*. So, the Gaussians will have be parameterized by a mean vector μ and covariance matrix Σ. The posterior is then defined as:

$$P(y|x) = \frac{N(x|\mu\Sigma)P(y)}{P(x)} \tag{3}$$

$$\text{where } P(x) = \sum_i P(y_i)N(x|\mu_{y_i}, \Sigma_{y_i}) \tag{4}$$

However, computation of Σ_{y_i} is a non-trivial task, so we can take a Naive approach of assuming all of our features are independent, that is:

$$\Sigma_{y_i} = I_n \tag{5}$$

An immediate potential downside to Naive Bayes is this naive assumption of independence of features, likely this assumption is violated in our data set, especially in the uni-gram case. For example words such as "Escherichia" and "coli" are likely highly correlated. However, there is empirical evidence to suggest that although Naive Bayes is optimal only when the independence assumptions holds true, Naive Bayes can still yield optimal results when evaluated by misclassification rate (Domingos and Pazzani, 1997). With this in mind, the Naive Bayes classifier should still yield reasonable results, or at least provide a baseline for the solving the *test_performed* (binary) classification problem.

3.2 Logistic Regression

Logistic Regression is an extension of linear regression that models the probability for categorical response variables. In our case, these will be the labels for each of our desired target variables *test_performed, test_outcome*, and *organism_name*. Observe that a normal linear model would not be appropriate for this task given that range for the output of linear regression is the real number line. Given our categorical outputs for each target, unless we apply some transformation to the output, this would lead to nonsensical results. So, with logistic regression we instead model the probability (log-odds) of each response. One assumption that still holds true is that we assume that the features are linearly related. In the binary case, the logistic model is defined as:

$$\log \frac{P(Y=1)}{1-P(Y=1)} = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_N \tag{6}$$

The logistic model with N features will have $N+1$ learned parameters $\{\beta_0, \beta_1, \cdots, \beta_N\}$ which we will denote as θ. We can retrieve $P(Y=1)$ from (6)

$$\log \frac{P(Y=1)}{1-P(Y=1)} = \theta^T \boldsymbol{x} \tag{7}$$

$$P(Y=1|X,\theta) = \frac{\exp(\theta^T \boldsymbol{x})}{1+\exp(\theta^T \boldsymbol{x})} = \frac{1}{1+\exp(-\theta^T \boldsymbol{x})} \tag{8}$$

In order to learn the parameters θ, we want to find θ such that we maximize the conditional log-likelihood of the training data:

$$\log \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i},\theta) = \sum_{i=1}^{n} -(1-y_i)\theta^T \boldsymbol{x_i} - \log(1+\exp(-\theta^T \boldsymbol{x_i})) \tag{9}$$

Unfortunately, (9) has no closed form solution, so we will estimate the parameters via numerical computation. This binary logistic regression will work for evaluating *test_performed*, however for the other two target variables *test_outcome* and *organism_name* we will need to extend the logistic regression to a multinomial setting. For K possible classes for the target variable, we run $K-1$ binary logistic regression models as outlined in (6), where one outcome is chosen as a baseline variable. That is:

$$\begin{aligned}
&\log \frac{P(y_i=1|\boldsymbol{x_i})}{P(y_i=K|\boldsymbol{x_i})} = \boldsymbol{\theta_1 x_i}\\
&\log \frac{P(y_i=2|\boldsymbol{x_i})}{P(y_i=K|\boldsymbol{x_i})} = \boldsymbol{\theta_2 x_i}\\
&\cdots\\
&\log \frac{P(y_i=K-1|\boldsymbol{x_i})}{P(y_i=K|\boldsymbol{x_i})} = \boldsymbol{\theta_{k-1} x_i}
\end{aligned} \tag{10}$$

The probability for each individual class can be computed by computing the probability for each of the $K-1$ binary logistic regressions as outlined by (8). For each observation, it will assigned to the class in which returns the highest probability. One possible downside to using (multinomial) logistic regression is the number of parameters that need to estimated. Given the large number of covariates (count vectorized *result_full_description*), lets call this r, we will need to estimate $(r+1)\times(K-1)$ parameters. Given the sparsity of the data set we are dealing with, it may be difficult to properly pick up the signals. One way we can tackle this issue is through regularization.

3.3 Regularization

Given the nature of the bag of words approach, we are introducing a high number of features, many of which are very sparse. Therefore, our models will be very complex which could pose the risk of overfitting our data. One method we can use to tackle this is problem is through Regularization. Regularization is performed by adding an additional regularization term to the loss function that we are optimizing. There are three common approaches to regularization: *Lasso*(L1), *Ridge*(L2) and *Elastic-Net*. The three regularization methods differ in the regularization term used when optimizing the loss function. For Logistic Regression, we are trying to maximize the condition log-likelihood of the training data (9), this is equivalent to minimizing the negative log-likelihood.

Lasso Regularization (L1)

$$\hat{\theta}_{lasso} = \arg\min_{\theta} \left[-\sum_{i=1}^{n} -(1-y_i)\theta^T \boldsymbol{x_i} - \log(1+\exp(-\theta^T \boldsymbol{x_i})) + \lambda \sum_{j=1}^{p} |\theta_j| \right] \tag{11}$$

Ridge Regularization (L2)

$$\hat{\theta}_{ridge} = \arg\min_{\theta} \left[-\sum_{i=1}^{n} -(1-y_i)\theta^T \boldsymbol{x_i} - \log(1+\exp(-\theta^T \boldsymbol{x_i})) + \lambda \sum_{j=1}^{p} \theta_j^2 \right] \tag{12}$$

Elastic-Net Regularization

$$\hat{\theta}_{elasticnet} = \arg\min_{\theta} \left[-\sum_{i=1}^{n} -(1-y_i)\theta^T \boldsymbol{x_i} - \log(1+\exp(-\theta^T \boldsymbol{x_i})) + \lambda_1 \sum_{j=1}^{p} \theta_j^2 + \lambda_2 \sum_{j=1}^{p} |\theta_j| \right] \tag{13}$$

All three methods of regularization apply a penalty parameter λ to the norm the parameters, L1 norm for the Lasso and L2 norm for Ridge. Elastic Net regularization combines both the L1 and L2 norm. The intuition behind this is given a model with very low bias, if we are fitting our training data too well, we can apply some penalty to the parameters to make it less perfect. The penalty parameter λ determines how significant the penalty should be, this is hyper parameter that needs to be tuned. Too small of a penalty and we may not address the overfitting problem, and too high of a penalty can swing the problem in the other direction, underfitting. Lasso regularization has an additional ability to shrink some coefficients to zero, which can be interpreted as feature selection. That is, L1 regularization can help remove useless variables.

3.4 Metrics

Choosing good metrics is important not only because they are necessary to evaluate model performance, but also because they are used to tune our model during cross validation. When choosing which metric to use, it is imperative we keep in mind the problem we are trying to solve, as well as the data set we are working with.

3.4.1 Common Metrics: Accuracy and F-Score

A common and very intuitive metric to assess the performance of our models is accuracy, that is the sum of all true positives and true negatives over the total number of observations. For multi class classification problems such as ours, it often the case that it is much more difficult to correctly classify the smaller classes due to the lack of training data. Since our data set faces a severe class imbalance, it is possible that we will get misleading results if we use accuracy as our performance measure. This is because that accuracy only takes in account how many predictions are correct, and does not take into account which class the correctly predicted samples are from. It is possible for our model to achieve high accuracy if we correctly predict the larger classes and fail to predict the smaller ones.

F-score is another common measure and provides some advantages over accuracy. Firstly, since F-score is the harmonic average between precision and recall, it will punish the model for over predicting the larger classes. Furthermore, it also give us flexibility and deciding how much we want to weigh the importance of precision and recall. For test performed, false negatives are much more undesirable relative to false positives, so, we can choose to weigh the recall more significantly. Furthermore, since we are dealing with a multi class classification problem, we take the F-score of each individual class and average them. We can choose to either weigh all classes equally, or even weigh the smaller classes more significantly to deal with the class imbalance.

$$F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{14}$$

3.4.2 Inversely Weighted Metrics

For our models, we want to more heavily punish incorrectly classifying the smaller classes, likewise we want to lessen the reward of correctly classifying the larger classes. We can accomplish this task by calculating the F-score and Accuracy more each individual class, then take an inversely weighted average. That is, the larger the class, the less weight we assign to it, and vice versa. The weight for class i is:

$$w_i = z \times N/n_i, \tag{15}$$

where n_i is the sample size for class i and z = 1/$\sum_i N/n_i$ is the normalizing constant. Then the overall F-score is the weighted average of the F scores of each class:

$$F_{model} = \sum_i w_i \times F_i, \tag{16}$$

where F_i is the F-score for class i.

3.5 Support Vector Machine

Support Vector Machines is a supervised learning algorithm that aims to draw a separating boundary between classes. For binary classification, suppose the data is linearly separable, then there is in fact many such boundaries that exist such that the data can be perfectly separated. SVM aims to find the boundary that maximizes the margin between the boundary and the training samples. That is, it aims to find a decision boundary that is as far from the data points as possible. Thus, we aim to solve this optimization problem:

$$\max_{w,b} \frac{2c}{||w||} \tag{17}$$
$$s.t.\ y_i(w^T x_i + b) \geq c,\ \forall i$$

where c is a scalar, if we set $c = 1$, it simplifies (17) without changing the optimization problem.

$$\max_{w} \frac{2}{||w||} \iff \min_{w} \frac{1}{2}||w||^2 \tag{18}$$

We write in a Lagrangian form:

$$\begin{aligned}
L &= \frac{1}{2}||w||^2 - \sum_i \alpha_i[y_i(w \cdot x_i + b) - 1] \\
\frac{\partial L}{\partial w} &= w - \sum_i \alpha_i y_i x_i = 0 \implies w = \sum_i \alpha_i y_i x_i \\
\frac{\partial L}{\partial b} &= \sum_i \alpha_i y_i = 0 \\
L &= \frac{1}{2}(\sum_i \alpha_i y_i x_i) \cdot (\sum_i \alpha_i y_i x_i) - (\sum_i \alpha_i y_i x_i) \cdot (\sum_i \alpha_i y_i x_i) \\
&- \sum_i \alpha_i y_i b + \sum_i \alpha_i \\
&= \sum_i \alpha_i - \frac{1}{2}(\sum_i \alpha_i y_i x_i) \cdot (\sum_i \alpha_i y_i x_i) \\
&= \sum_i \alpha_i - \frac{1}{2}(\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j)
\end{aligned}$$

This is a quadratic optimization problem. Having calculated all parameters, for a new observation u, all we need to compute is the dot product of u and all the x_i to classify u. If the following equality holds

$$\sum_i \alpha_i x_i \cdot u + b \geq 0, \tag{19}$$

then u is classified as positive, and negative otherwise. For data that is not linearly separable, we can apply a kernel trick but applying a non-linear transformation on x_i and u such that the data points will be linearly separable in the new transformed space.

3.6 Random Forest

Random Forest is a ensemble technique that utilizes bootstrap aggregation of many decision trees. By ensembling many decision trees, the Random Forest algorithm remedies the overfitting tendencies of individual decision trees.

3.6.1 Decision Tree

A decision tree is a supervised learning algorithm that is structured like a flowchart. A decision tree decision tree is comprised of three types of nodes:

- Root Node: It is the first node in the tree that contains the entire training set.
- Internal Node: It corresponds to a feature, on which a tree splits into two. It it also known as decision node.
- Leaf Node: It is a terminal node in the tree that corresponds to a class label

The decision tree is created by first selecting the most useful feature and placing it at the root node. The most useful feature is the one that splits the data into two most homogeneously. Usefulness is determined by how *pure* a tree is. A common measure for impurity is Gini index:

$$Gini = 1 - \sum_{i=1}^{K} p_i^2, \tag{20}$$

where K is the number of classes, and p_i is the probability of the class i. p_i is calculated as the number of observations in class i divided by total number of observations at each node. The Gini index is low or pure when observations from one class dominates.

A stump is the simplest form of a decision tree, where the root node is the sole internal node. Just having one internal node is not often sufficient for the model to work well in terms of prediction errors. To have more than one internal node, the above process is repeated at each of the node from the previous nodes. That is, at each node, the most useful feature is chosen again such that the chosen feature results in the least impure subset of data. The decision to stop splitting can be based on a number of factors or hyperparameters that are chosen beforehand:

- Maximum depth of tree
- Minimum samples to split internal node
- Minimum samples in leaf node
- Minimum impurity decrease
- Minimum impurity to split

Decision Tree Pruning As the depth of a decision tree increases, the more prone it is to overfitting. By increasing the depth of tree, training error is reduced, however this may come at the cost of catching too much the noise in the training data which could cause the model to not perform well when subjected to new data. In other words, the decision tree learns the training data too well, perhaps yielding leaf nodes with a single observation, as such the tree would not be able to classify data well that varies from the training set. That is, inherently decision trees are low bias but high variance.

3.6.2 The Random Forest Algorithm

The process of creating a Random Forest involves creating M new training sets by sampling n observations from the training set uniformly and with replacement, and then fitting M models. Naturally, this will involve duplicate observations in some of the training sets. Bagging reduces variance by averaging many noisy but relatively unbiased models. In addition, averaging out the results of many models yields a more stable or smooth results that is less prone to overfitting. The final prediction is made via averaging the result of each model in the case of regression, or through a voting process in the case of classification.Random forest is an algorithm which builds a collection of decision trees and trains them using the bagging method. Random forests help correct for decision trees' tendency of overfitting by averaging the result of a large number of trees. In the case of classification, the random forest algorithm uses a voting procedure to predict class label of an observation, that is taking the mode class label of the individual decision trees. A key component of random forests is that it adds a degree of randomness. Unlike decision trees, rather than selecting the most useful feature to split on, the most useful feature is selected from a random subset of the total features. Since each tree in the random forest is identically distributed, for M trees each with variance σ^2 the average variance is:

$$\rho\sigma^2 + \frac{1-\rho}{m}\sigma^2 \tag{21}$$

where:

- ρ is the positive pairwise correlation
- σ^2 is the variance
- M is the number of trees

If M is sufficiently big, the second term disappears, and by randomly selecting subsets of features we can reduce the correlation between trees, hence reducing overall variance.
The random forest share a lot of the same hyperparameters of decision tree, that is for each tree in the random forest, it can be tuned via the hyperparameters outlined in section 5.

Other hyperparameters that can be altered to tune the random forest algorithm are:

- number of trees
- maximum features

As the names imply, the number of trees is the number of decision trees that populate the random forest, and the maximum features is the maximum number of features that the random forest will consider splitting on. Although it is true that Random Forest often produces good results, since each decision tree is trying to maximize purity on each split, it is possible for it to miss the very small classes. That is that a split can be quite pure even if it misclassifies a class that contains very few observations. This is particularly relevant in our data set, *test_outcome* is highly unbalanced, and *organism_name* contains many very small classes.

3.7 Adaboost

Boosting is a similar concept to bagging, it combined a series of weak learners in order to produce a single strong learner. AdaBoost algorithm sequentially produces decision stumps, and combines the prediction of each stump for its final classification. AdaBoost differs from random forests in the sense that each tree is not given equal vote in the final classification, and the order in which the stumps are produced matters. At each time step in which a tree is produced, the observations that were misclassified in the previous model are given more emphasis, or in words weighted more heavily. Treating the weights as a distribution, a new training set of the same size is selected with replacement. In other words, the higher weight makes the incorrectly classified samples more likely to selected to train the next decision stump. The greater weight means that if misclassified, the tree would be more heavily penalized, as the error is a function of the weights. The penalization, or error of the tree determines the final voting power the tree has in the final classification. This process is repeated until M decision stumps are created, in which case for each observation, the sum of the tree's amount of say is calculated for each class label, and the class label with the higher score is the predicted class. We define:

- w_i^t the weight given to observation i for tree t
- $A = \begin{cases} +1 & y_i \neq f_t(x_i) \\ -1 & y_i = f_t(x_i) \end{cases}$ where y_i is the true classification and $f_t(x_i)$ is the predicted classification
- $I(*) = \begin{cases} 1 & * \text{ is true} \\ 0 & * \text{ is false} \end{cases}$

The algorithm is as follows:

1. Find best feature based on some impurity measure such as Gini index and create decision stump using the selected training set (selected based on weights of each observation, at t=1 the entire training set is used as $w_i = \frac{1}{N} \ \forall \ i$
2. Compute the $\epsilon_t = \sum_{wrong} w_i^t$ the model error of decision stump t
3. Compute $\alpha_t = \frac{1}{2}\log(1 - \epsilon_t/\epsilon_t)$, the amount of say of decision stump t
4. Compute new weight of each observation $w_i^{t+1} = w_i^t \exp(A\alpha_t)/z \ \ \forall \ i$ where $z = \sum_{i=1}^N w_i^t$ (a normalization factor)
5. Repeat until M stumps are created

After the M trees are created, the class label for each observation is determined based on the votes given to each class, where the class with greater votes is the predicted class. For each observation i, votes are determined by:

$$Votes_{class} = \sum_{t=1}^{m} \alpha_t I(Class = f_t(x_i)) \tag{22}$$

3.8 Deep Learning Method

There are two main downsides with the machine learning methods above. First, using the Bag-Of-Words approach to vectorize the result descriptions will create highly sparse feature matrices, and since only the count of each word is used the meaning of each word is lost. We propose a solution to this problem by using Word Embeddings rather than bag of words to numerically represent each word. The second downside is that the sequence of the result descriptions are not taken into account, that is the order of the text is lost. An immediate example of when the sequence of the text is important are negations, e.g. *not found*, *not positive*, etc. Another example is for result descriptions that contain corrections. Since the result descriptions are append only, if there were incorrect observations they would not be removed and would only be corrected later on in the description. So, there could be situations in which the text that appear later in the description hold more value. Using a Recurrent Neural Network to perform classification, will allow each word of the result description to be fed in sequentially. By using this approach, we hope to be able to improve classification performance. In additional, also have the ability to obtain every organism name and their corresponding test outcomes within a result description, if a result description contains more than one organism.

3.8.1 Word Embeddings

Like Bag-of-Words, Word Embeddings are a vectorized representation of words. That is, Word Embeddings is a method to transform each word in a sentence into a vector (numerical) representation. Word Embeddings are trained such that words used in similar contexts will have similar vector representations, and words not often found together will have dissimilar representations. That is, for some measure of distance, words that are similar will be relatively close, and words that are dissimilar will be far apart. An immediate advantage this presents is that the meaning and context of the words will be accounted for.

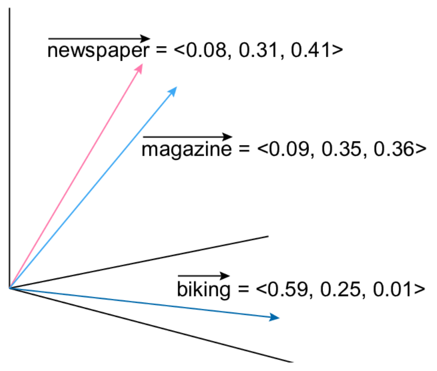

$\overrightarrow{\text{newspaper}}$ = <0.08, 0.31, 0.41>
$\overrightarrow{\text{magazine}}$ = <0.09, 0.35, 0.36>
$\overrightarrow{\text{biking}}$ = <0.59, 0.25, 0.01>

Figure 6: Example of 3-dimensional Word Embeddings

For our project, we will be using pre-trained word embeddings. Popular pre-trained word embedding models include Word2Vec, GloVe and fastText. The embeddings that we will be using come from BioWordVec (Zhang *et al.*, 2019), 200 dimensional word embeddings obtained using the fastText algorithm trained on a corpus composed of literature from PubMed and MeSH. Pre-trained embeddings are a set of weights (vectors) that can be mapped to each word in our result description in the first layer of our Neural Network. For words that are in our result descriptions but not in the pre-trained embeddings, we will either map the word to a random vector or a vector to denote unknown words. As a note, after these weights are mapped to each word, we can further train the model to tweak the weights such that the vector representation for each word is learned based on our data, using the pre-trained embeddings as a starting point.

3.8.2 Recurrent Neural Network

In order to take into account of the sequence of text, we will be using a Recurrent Neural Network (RNN) as our classifier. The architecture of a vanilla RNN is as follows:

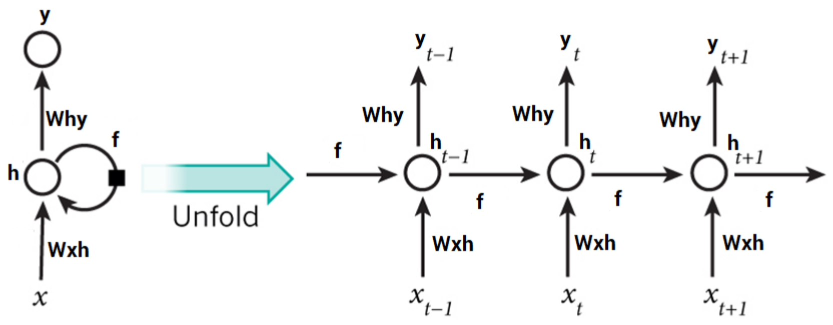


Figure 7: RNN architecture (Gupta, 2017)

For each word x_t in the result description, it is inputted and stored as a hidden state h_t

$$h_t = f_w(W_{hh}h_{t-1}, W_{xh}x_t) \tag{23}$$

where

- f_w is some activation function (e.g. $\tanh$)
- W_{hh} and W_{xh} are the parameters of the model, they are the same for each word

The hidden state computed from each word takes into account the inputted word and the previous hidden state. This is how an RNN is able to take into account the the sequence of text. Figure (7) above is an example of many-to-many RNN where every input has an output. In our case, for the purpose of finding *test_outcome* we will be using a many-to-one RNN, where only a single output is returned after all the words are inputted.

$$y_i = W_{xh}h_T \tag{24}$$

A common issue with RNNs is the inability to capture long term dependencies, that is earlier inputs are often forgotten (Hochreiter, 1998). This occurs due to the nature of how the parameters are learned and is known as the "vanishing gradient problem". Two common RNN architectures that combat this issue is LSTM and GRU, the latter of which we will implement in our classifiers. GRU adds two additional gates: the update gate z_t and the reset gate r_t. These gates contain their own set of parameters that need to be learned, the update gate tweaks how much of the previous hidden state to include and the reset gate is used to drop previous information.

$$r_t = f_r(W_r x_t, U_r h_{t-1}) \tag{25}$$

$$z_t = f_z(W_r x_t, U_r h_{t-1}) \tag{26}$$

$$h_t = z_t * h_{t-1} + (1 - z_t) * f_w(W_{hh}(r_t * h_{t-1}), W_{xh}x_t) \tag{27}$$

At a high level, GRU adds additional parameters that allow previous information to be dropped and decide how much of the previous output to include in the next hidden state. The ability to handle sequential data will enable us to reach the eventual goal of being able to find multiple organisms and their corresponding test outcomes. If we do not take into account the sequence of the text, we would have no way of associating which test outcome corresponds with which organism name.

3.9 Active Learning

While we defined labeling multiple organisms and its corresponding outcomes in a lab test result as the new objective that is more aligned with BCCDC's goals and identified a recurrent neural network as an appropriate method to achieve it (Section 3.8), we run into a serious issue: lack of training data (we only have 20 or so). Building a training data set out of a large amount of unlabeled data for a machine learning model is not a simple problem, especially when working at scale. In particular, one needs to think about how to query new data points to label and how much training data is needed for a machine learning model to provide a desired level of performance.

Active learning is a framework, in which a machine learning model selects which new data points to be added to its training data set Settles (2009). It is an iterative process as shown in Figure 8. For a given pool of unlabeled data U and an initial set of labeled/training data $\mathcal{L}$, a machine learning model is trained on $\mathcal{L}$ and then used to provide a level of uncertainty for each data point in $\mathcal{U}$ by some uncertainty scoring function. Then by some query mechanism, a pre-specified number of data points is drawn from $\mathcal{L}$ to be labeled by a human annotator.

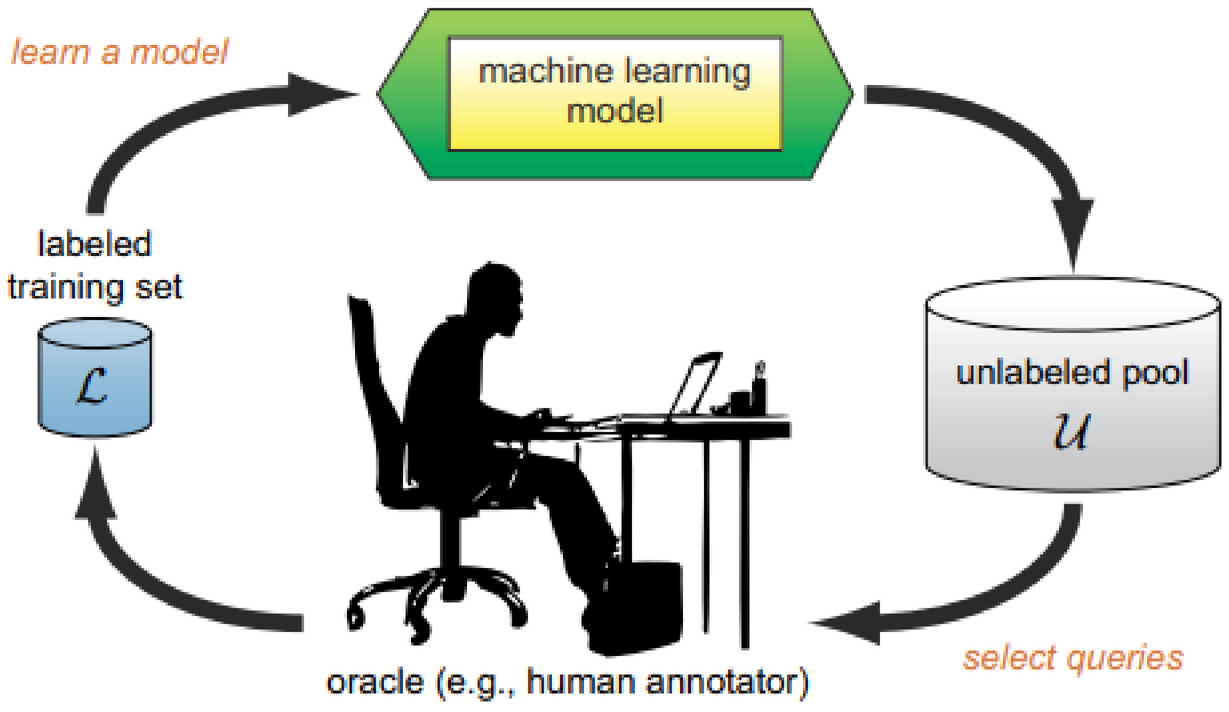


Figure 8: The (pool-based) active learning cycle (Settles, 2009)

To prove the concept of active learning, we use RF as the machine learning model with test_outcome as the target on the HCV dataset. As for the uncertainty scoring function, we use the mean predicted class probabilities. For querying new data points, we simply take the data points in decreasing order of the uncertainty score. We compare active learning and *random* learning (i.e. queried randomly), and results are provided in Section 4.3. Other practical aspects of active learning are discussed as future work or considerations in Section 5.2.

3.10 Hyperparameter Optimization

Hyper-parameters are configuration variables of a machine learning model that must be chosen prior to fitting or training the models. In other words, these variables are not updated or *learnt*; they determine the structure of a model. For instance, the number of trees and the maximum number of features are the hyperparameters for RF (Sec. 3.6), and the choice of a kernel (linear vs. non-linear) is a very important hyperparameter for SVM (Sec. 3.5). Hyperparamter tuning of a machine learning model is crucial for performance.

Formally, let $\mathcal{X}$ be the space of all the hyperparameters X of a machine learning model and $f \in \mathbf{R}$ be an objective function that we wish to maximize (or minimize). We assume that f does not have a simple closed form but can be evaluated at some realized value x of X. Our objective is to find the best set of values $x^* \in X$ such that f is maximized.

3.10.1 Classical Approaches

The classical approaches to hyperparameter optimization are grid search and random search. In grid search, each hyperparmeter is assigned a set of possible values to create a search space (a subset of $\mathcal{X}$), where the researcher believes the optimal set of hyperparameter values x^* lies. Then every combination is evaluated, and the combination that gives the maximum output is chosen. While this approach can give you a guarantee, it might be too costly when the search space is big. Random search alleviates this issue by randomly sampling in the search space until some criterion, which is set by the researcher, is met. However, this approach may not give you a guarantee that the set of hyper-parameter values chosen by random search is the optimal one within the search space. In practice, the latter approach and its variants are shown to work effectively. One shortcoming is that both of these approaches do not scale well as the number of hyperparameters increases; the cost grows exponentially.

3.10.2 Bayesian Optimization

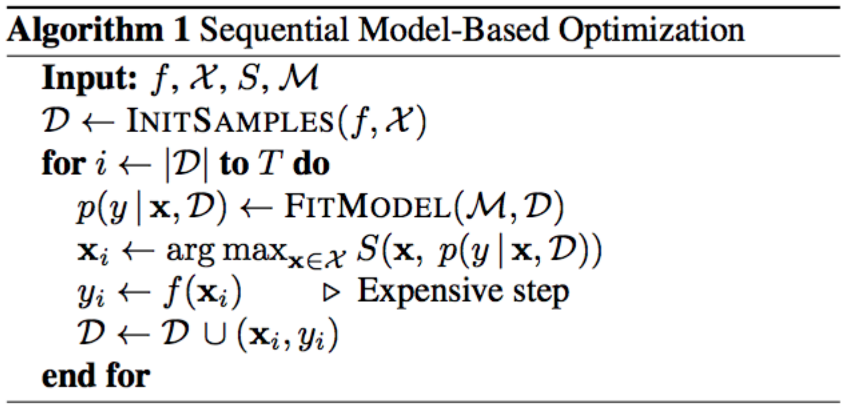
Bayesian optimization is a powerful tool for solving the hyperparameter optimization problem when it is costly to evaluate f and/or the search space is too big. It is a class of sequential model-based approaches that builds on past evaluations to choose the next optimal set of hyperparameter values that would maximize f. This is the main difference from the classical approaches, which do not consider the history of evaluations. To discuss how Bayesian optimization, we introduce three more terms:

- D: a set of historical evaluations $(x_i, y_i)_{i=1}^{n}$, where $y_i = f(x_i)$ and $x_i \in \mathcal{X}$.
- M: a probabilistic model that constructs the predictive (posterior) distribution of y, $p(y|D, M)$. This predictive distribution captures uncertainty associated with finding a new sample for evaluation in the surrogate step.
- S: a cheap surrogate for f. It is used to find the next optimal sample to evaluate based on D by balancing between exploring new areas in $\mathcal{X}$ and exploiting the explored areas D. A commonly used S is the expected improvement function, EI (Jones *et al.*, 1998):

  $$EI(x) = \int_{f_{best}}^{\infty} (y - f_{best}) p(y|x, D) dy, \tag{28}$$

  where f_{best} is the most optimal objective function value found so far. For a new $x \in \mathcal{X}$, $EI(x)$ is the expected improvement over f_{best}.

A general algorithm for Bayesian optimization for tuning hyperparameters is provided in Fig 9.



```
Algorithm 1 Sequential Model-Based Optimization
  Input: $f, \mathcal{X}, S, \mathcal{M}$
  $\mathcal{D} \leftarrow$ INITSAMPLES($f, \mathcal{X}$)
  for $i \leftarrow |\mathcal{D}|$ to $T$ do
    $p(y \mid \mathbf{x}, \mathcal{D}) \leftarrow$ FITMODEL($\mathcal{M}, \mathcal{D}$)
    $\mathbf{x}_i \leftarrow \arg\max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}, p(y \mid \mathbf{x}, \mathcal{D}))$
    $y_i \leftarrow f(\mathbf{x}_i)$        ▷ Expensive step
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{x}_i, y_i)$
  end for
```

Figure 9: Algorithm for Bayesian optimization (Ian Dewancker and Clark, 2015). T is the maximum number of iterations for finding x^*; it represents the maximum time or resources allowed.

Currently, there are four well-known Bayesian optimization software: SPEARMINT, MOE, HYPEROPT, and SMAC. These software differ in how they model the predictive/posterior distribution. Note that in all of these software, they use the same S: EI. For a complete review of the four software and details, please see Ian Dewancker and Clark (2015). For the project, we use HYPEROPT for its good compatibility with Python (Komer *et al.*, 2014). We implemented hyperparameter optimization via HYPEROPT in the existing pipeline for machine learning classifiers. Furthermore, to accommodate hyperparameter optimization, a significant effort was made on re-coding the feature engineering step.

4 Results

In this section, we will go over the results of our classifiers. More specifically, we will go over how including additional features has improved the baseline machine learning classifiers. For all the machine learning methods, we use 5-fold cross validation to decide which classifier to used for the final classification. The results below are based on a 80-20 train-test split, that is 80 % of the data is used for training the classifier and 20 % is used for testing. The deep learning results are based on the HCV data set and only uses 3000 data points for training.

4.1 Machine Learning Results

For *test_performed*, applying the machine learning algorithm, using just the bag-of-words obtained from the result description we obtain Accuracy and F2-score in excess of 0.990. When we include additional features, the classifier performance improves to an accuracy and F-2 score of 0.999.

Test Performed	Bag of Words	Bag of Words + Number of Observations + Test Code + Metamap + Candidates
Accuracy	0.996	0.999
F2-score	0.990	0.999

Figure 10: Test Performed Classification Results

Similarly for *test_outcome*, the baseline bag-of-words approach already provides good results in terms of accuracy and inversely weighed F1-score and accuracy. By including additional features, we again able to improve the performance across all metrics over the baseline approach.

Test Outcome	Bag of Words	Bag of Words + Number of Observations + Test Code + Metamap + Candidates
Accuracy	0.992	0.994
Inversely-weighted accuracy	0.960	0.974
Inversely-weighted F1-score	0.944	0.961

Figure 11: Test Outcome Classification Results

For *organism_name*, it is clear that using inversely weighted metric is much more appropriate for the evaluating performance. Although the baseline bag-of-words approach has high test accuracy, we can see that that many of smaller classes are missed as denoted by the inversely weighted metrics. When we included the additional features we were able to improve the inversely weighted accuracy from 0.636 to 0.866 and the inversely weighed F1-score from 0.664 to 0.848.

Organism Name	Bag of Words	Bag of Words + Number of Observations + Test Code + Metamap + Candidates
Accuracy	0.947	0.956
F-2 Score	0.790	0.874
Inversely-weighted accuracy	0.636	0.866
Inversely-weighted F1-score	0.664	0.848

Figure 12: Organism Name Classification Results

4.2 Deep Learning Results

We prototyped the Recurrent Neural Network on *test_outcome* of the HCV data set. The HCV data set guarantees a single *test_outcome* for each observation, which allowed us to replace the organism name within each result description with a "TARGET" token using a rule based approach. Neural Networks tend to perform better when given a large data set, and even though we only used about 3000 training points the Neural Network was able to outperform the Random Forest across all metrics.

Test Outcome	Deep Learning (RNN)	Machine Learning (Random Forest)
Accuracy	0.994	0.993
F1-Score	0.929	0.924
Inversely-weighted accuracy	0.890	0.789
Inversely-weighted F1-score	0.890	0.858

Figure 13: Test Outcome (HCV) Classification Results

4.3 Active Learning Results

We prototyped an active learning framework using the HCV data set. Starting with 100 training data points and iteratively adding 100 additional training points, we can compare the results of the query mechanism versus random sampling. By using a query mechanism, we were able to achieve the same results (> 0.95 accuracy and F Score) using just over 2000 training points, whereas the random sampling required over 35000 training points.

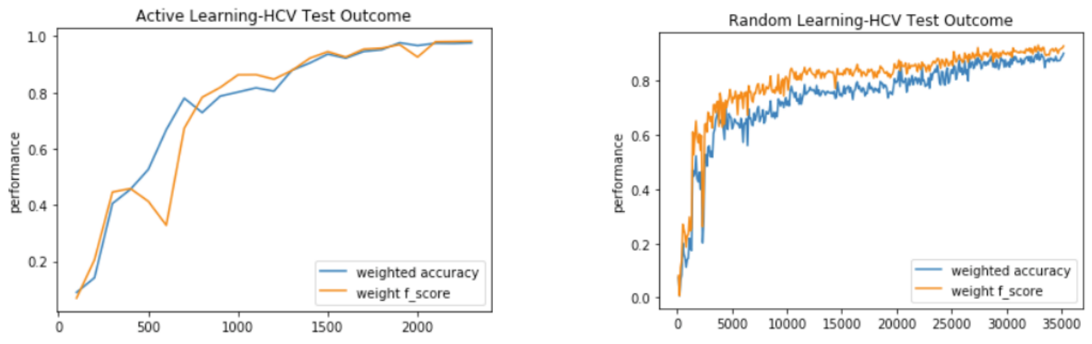


Figure 14: Performance of Active Learning vs Random Learning

If we dive deeper into why exactly the active learning approach is outperforming random learning, we can see that it is precisely due to the query mechanism. Due to smaller classes being more difficult to predict, the underlying classifier is more uncertain when predicting their labels. So, the active learning model queries the smaller classes for manual labeling. Very early on, the active learning approach begins to stop querying the large classes (positive, *missing) and seeks out the smaller ones (negative, indeterminate). Whereas with random sampling, as expected it queries the data uniformly. Thus, the active learning approach is able to quickly increase its performance on the smaller classes, and in turn increase its overall performance much more quickly as compared to random learning.

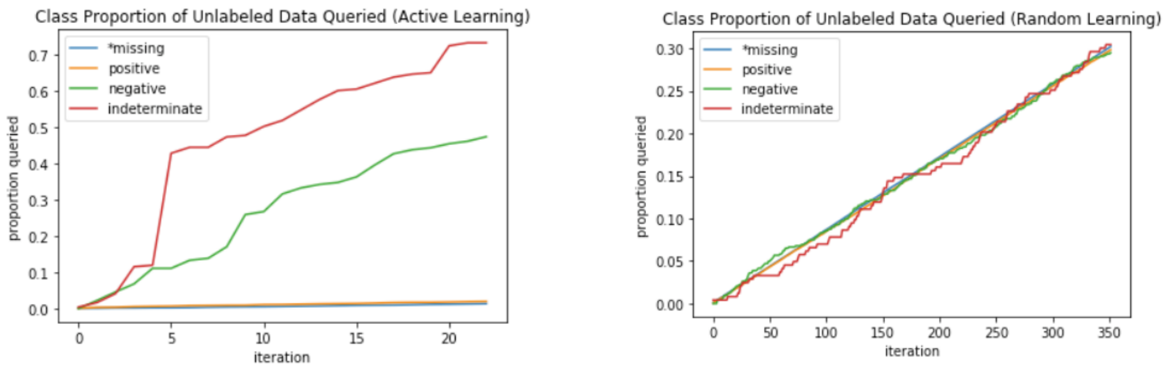
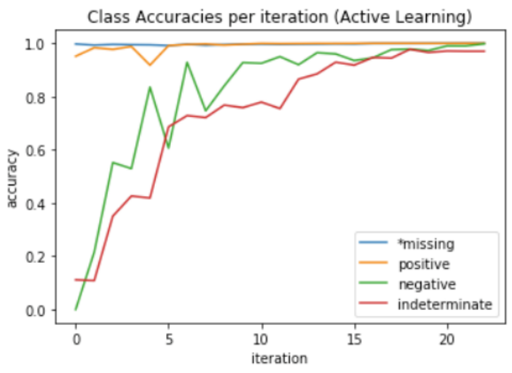


Figure 15: Querying of Active Learning vs Random Learning



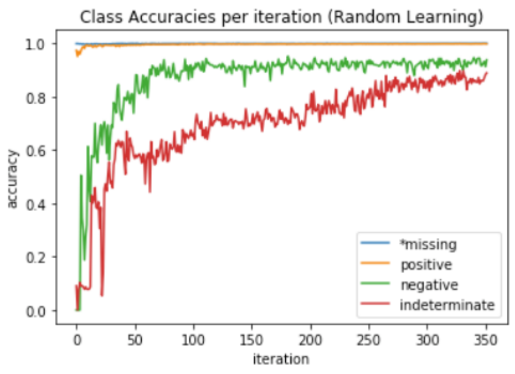


Figure 16: Class Performance of Active Learning vs Random Learning

5 Conclusion

5.1 Conclusion

Our major contribution to this project consists of three parts. The first part is modification on the existing machine learning pipeline. Building upon the work of previous DSSG fellows, we introduced more data preprocessing steps in order to remove components of the text that can potentially results in misclassification. Firstly, we remove stop words to reduce the sparsity of the word vectors. Secondly, we remove all sentences containing the key phrase "previously reported as" to remove the part of the description field that contradicts to the correct result. Moreover, based on suggestions from the BCCDC's domain experts, we included more features in training classifiers, such as number observations and lab technician ID (explained in section 2.1). Furthermore, since it is necessary to tackle the severe class imbalance in this study so that the trained classifier does not only focus on correctly classifying the large classes, we use inversely-weighted accuracy or F1-score as a performance metric in both cross validation and comparing classifiers.

After the changes above were integrated into the pipeline, the performance of the best trained classifier has improved for all of the *test_performed*, *test_outcome* and *organism_names* classification tasks. Up to this stage of development, the classic machine learning approach seems to have achieved an ideal performance for classifying the *test_performed* target.

Besides changes in preprocessing, feature selection and metrics, we also experimented hyperparameters optimization with the existing pipeline, which is expected to optimize classifiers to more extent than those with fixed hyperparameters. However, our trials have shown that hyperparameters optimization turns out to take significantly longer time and does not improve the classification accuracy by much. Therefore, we did not implement it into the pipeline.

Although we did not build a complete pipeline for the deep learning approach, exploratory work has shown (in section 4.2) that deep learning has indeed improved the performance of the best trained classifier compared to regular machine learning. Since word embedding takes in account the ordering of words within a sentence while bag of words does not, it seems to be more suitable for identifying multiple organism(s) and corresponding test outcome(s) than bag of words. Future work recommendation of this branch of the project is described in section 5.2.

Later in the project as mentioned in section 3.9, we implemented a prototype of the active learning framework to account for the lack of labelled data for classifying *organism_names*. With active participation from the BCCDC's experts, active learning can achieve training the classifier and building the labelled dataset simultaneously. Results were shown promising that our proposed query mechanism can reduce the error rate more rapidly than random learning. Moreover, since the new objective requires all microorganisms along with their test outcome(s) labelled in order to apply any learning algorithm, which has to be done manually, active learning is the only feasible approach based on our knowledge, and will provide a larger fully labelled dataset for other learning approaches in the future.

5.2 Moving Forward

While our proposed methods showed promising results, there still remains much work to be done to turn the purposed solution into a viable product for the BCCDC. There are largely three components.

The first component is a method for identifying organism names in a laboratory test result. An adhoc method was implemented for this task for the HCV dataset, as we know apriori that there is only one organism of interest in the dataset. Fuzzy string matching and hard-coded abbreviations were sufficient to identify all the potential spellings of *Hepatitis C Virus*. This adhoc method would certainly not work at scale. As an alternative, we tried Metamap and found that it did not give a satisfactory result (30% accuracy on a small toy dataset). Our recommendation is a machine learning method that builds on Metamap (i.e. outputs from Metamap are taken as input for a machine learning model) in the active learning framework, as a training dataset is not available.

The second component is a method for determining a test outcome of each organism in a laboratory test result. As discussed in Section 3.8, a deep learning model is well-suited to solve this task and showed promising results for the HCV dataset. Due to the time constraint at the end, we did not have enough time to implement a pipeline for fine-tuning a deep learning model (hyperparameter optimization) and for incorporating additional features, such as test sub-type and technical ID. We suggest that additional features are concatenated just before the final output layer of a deep learning model rather than putting those as texts in a laboratory test result and using them in a word embedding. To extend our methodology to a case of multiple organisms, we use an iterative approach. For an organism in a lab test result, we replace its name with a target token, say _TARGET_, and replace the other organism names with another token, say _OTHER_. Then we feed that modified text to the deep learning pipeline. We repeat the same process for each of the organisms in the lab text result.

The third component is active learning. The first two components rely on active learning to efficiently create a labeled/training dataset for the two methods. While we showed the power of active learning on the HCV dataset, our implementation is not ready for practical use. In particular, there are several aspects of active learning that should be considered: multi-task active learning (since we now have two models to train), stopping criteria (when to stop labeling), and unreliable oracles (inevitable errors from human annotators). These issues are discussed in-depth in Chapter 7 of Settles (2012). In addition, to implement the active learning framework, an interactive interface is required for human annotators to query and label data points easily without having to write programming codes.

Accomplishing each of the components is not trivial. It would require an experienced machine learning scientist (NLP) and software engineer. There do exist two commercial products that may perform all of the above tasks: Amazon Sagemaker (https://aws.amazon.com/sagemaker/) and Prodigy (https://prodi.gy/). In a conference, we met a data scientist at WorkSafeBC and was informed that they use Prodigy for classifying customer complaints.

5.3 Social Good

We conclude the report with a remark on impact of our work on social good. Our work moves the BCCDC closer to replacing the manual labeling process with an automated system. Furthermore, We proposed the new, ambitious objective of identifying all the organism names in a lab test result and determining their test outcomes and described a potential solution to achieve it. The automated system would bring direct impact on population health. It would require less time and capital resources to label lab test results and likely reduce human-made errors. Consequently, it would decrease the economic burden on BCCDC, and most importantly, more accurate statistics and analyzes based on the labeled lab test results would be available to stakeholders in a shorter time. It would imply that health authorities could make more accurate and quicker decisions on urgent matters, such as disease outbreaks. In addition, with all the organism names and their test outcomes labeled, health authorities would have more granular data to look into matters relevant to population health.

References

Sizhe Chen, Kenny Chiu, William Lu, and Nilgoon Zarei. Classifying laboratory test results using machine learning, 2018.

Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.

Dishashree Gupta. Fundamentals of deep learning – introduction to recurrent neural networks, 2017. Last accessed 30 Dec 2019.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Michael McCourt Ian Dewancker and Scott Clark. Bayesian optimization primer, 2015. Last accessed 14 June 2019.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, pages 2825–2830. Citeseer, 2014.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

Burr Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

Yijia Zhang, Qingyu Chen, Zhihao Yang, Hongfei Lin, and Zhiyong Lu. Biowordvec, improving biomedical word embeddings with subword information and mesh. *Scientific data*, 6(1):52, 2019.