Final Project Evaluation Report - Machine Learning

Personalized Medicine: Redefining Cancer Treatment

Group No. 54

Aditya Adhikary
IIIT Delhi
aditya15007@iiitd.ac.in

Dhruva Sahrawat
IIIT Delhi
dhruva15026@iiitd.ac.in

1. Introduction

1.1 Problem Statement

In this project, we have an expert-annotated knowledge base where world-class researchers and oncologists have manually annotated thousands of mutations in genes. These mutations have been classified into 9 classes. Our goal is to find a machine learning algorithm that, when given the text, automatically classifies these genetic variations.

1.2 Motivation

A cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers). This interpretation of genetic mutations is being done manually, which is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature. Hence, we are motivated to use machine learning and natural language processing techniques in order to automatically classifies these genetic variations.

2. Related Work

2.1 Literature Survey

With a tremendous amount of literature now available in the biomedical field, a number of research experiments have been carried out to use text-mining approaches to automatically annotate genes, mutations, diseases and so on.

A similar study has explored the possibility of using a text mining system for associating a particular mutation with a disease. A number of other open-access tools have been created which apply NLP techniques on Biomedical literature for related problems: - *tmVar* is used for extracting a range of sequence variants in both protein and gene levels, and *TaggerOne* is a more general toolkit for biomedical named entity recognition, which also uses text-mining approaches for locating and identifying concepts such as diseases and chemicals in biomedical text.

2.2 Kaggle Data Challenge Scores

This is a recently completed challenge on Kaggle. The best observed scores (in multi-class log loss) for the public leaderboard section, which uses 76% of the test data, can be treated as the state-of-the-art results obtained so far on the given dataset. The best results achieved is around 0.11 and 2.03 on the private leaderboard.

3. Dataset and Evaluation

3.1 Data Description

There are two csv files and two text files, 1. *training_variants.csv* with 4 columns: Gene[String], the particular gene where the mutation took place; Variation[String], the nature of the mutation;

Class[String], manually classified assessment of the mutation; ID[Integer], used to link this file to the training_text file. The IDs are labelled from 0-3320.There are 9 Classes in total, labeled 1-9.
2. *training_text.txt*: Corresponding to each ID, there is an extensive description of the evidence that was used to manually label the mutation classes.
Similarly, there is a *test_variants.csv* file for the above, which doesn't have the Class values, which we have to predict using the *test_text.txt* file. Hence, the number of samples in the training set are 3321, and for the test set, 986. We are doing K-fold cross-validation on the training set and hence the number of samples in the validation set depends on the value of K, i.e. if K = 3 then the validation set in each iteration has approximately 1107 samples.

3.2 Preprocessing

We have used the *nltk* library to 1. Tokenize the documents to words and remove punctuations, 2. Stem the words using a snowball stemmer, 3. Remove commonly occurring stop words as well as common words found in literature, and stored this new training dataset in a file as a list of words for each sample. We do the same on the stage 2 test dataset before prediction.
We also tried *upsampling* for the classes with lower number of labels associated with them.

3.3 Feature Extraction Techniques

We have used the Bag-of-Words and Bigram/Trigram models successfully by using the *CountVectorizer* and *Tfidf* libraries to convert variable length text documents to sparse vectors before running any machine learning algorithm on them. We also used *Word2Vec* and *Doc2Vec*, which consist of shallow, two-layer neural network models that are trained to reconstruct linguistic contexts of words, thus producing word embeddings(texts converted into numbers). *Word2Vec* contains two distinct models, *Continuous Bag-of-Words* and *Skip-gram*. The first tends to predict the probability of a word given a context(a single word or a group of words). The input is the 1 hot encoded word, and there is a linear activation function between any layers.The weight between the hidden layer and the output layer(softmax activation) is taken as the word vector representation of the word. Skip-gram does the inverse, i.e predicts a word given a context. In *Doc2Vec*, instead of using just words to predict the next word, we also add another feature vector, which is document-unique. We also used *TruncatedSVD* on the tf-idf vectors, which uses SVD on them to perform dimensionality reduction. This preserves the most important semantic information in the text while reducing noise etc. We also encoded the 'Gene' and 'Variation' columns to our feature vector for better classification by using a one hot encoding and then latter using truncated SVD to reduce dimensionality.

3.4 Evaluation metrics

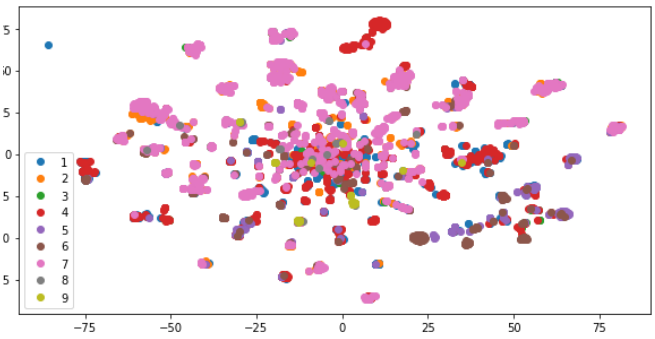
We have used *StratifiedKFold* with k =3 or 5 to cross-validate and *cross_validate_predict* to fit a model and predict on the training dataset. We have plotted *confusion matrices*, found the prediction *accuracies* and *log losses* for each model under evaluation. We also look at the *Precision*, *Recall* and *F-score* of each class, since the labels are imbalanced and we need more than accuracy as a metric. We select the model with the best performance during validation, i.e the best combination of log-loss and accuracy scores, having a balanced confusion matrix, and high precision and recall.

4. Methodology

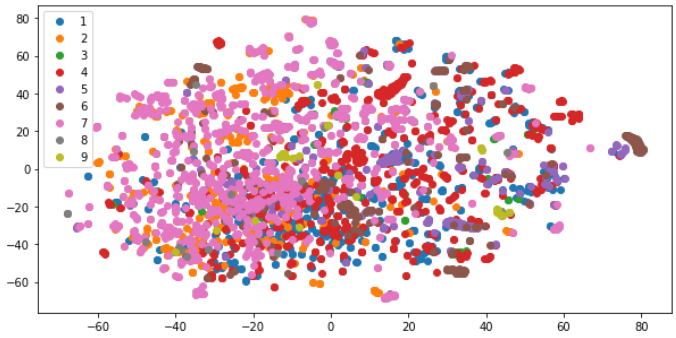
4.1 Visualization of Data and General Analysis

We did a few checks on the quality and type of data, from which it appeared to be entirely inseparable. The training data is highly unbalanced class-wise, and

classifiers almost never predict a sample to be in classes 3, 8 or 9.



Here we used TruncatedSVD and TSNE to plot the data points whose features are simple sparse matrices gotten from Tfldf.
After we use doc2vec and add gene and variation to our feature pool, we get the following representation of data points through TSNE.



As we can see the data points are not linearly separable for any of them. The doc2vec vector have less no of features and give better accuracy compared with normal Tfidf vectors.

Grid search takes too much time due to its brute-force approach, and hence hyperparameter tuning is difficult from a time perspective. Hence, we tried GridSearch for only some select models with better cross-validation metrics, such as RandomForest,logistic, SVM and XGB with modified feature space vectors of word2vec and doc2vec with gene and variation encoded into features.

The log-loss in the hold-out (test) set is showing to be better on the Kaggle submission than the cross-validation step on the training set, suggestive of underfitting and small number of training samples. The gap between our performance and the best possible performance is because we have not used any other Neural Networks based model (apart from LSTM) so far such as CNN or RNN.

4.2 Design Choices

We have used the following learning methods:
1. *Multinomial Naive Bayes* classifier, since it works well as a benchmark/ starting point and is suited for sparse matrices of the type the vectorizers output for the samples,

2. *Logistic Regression*, as it uses a one vs rest strategy for classification and serves as another essential base model,

3. *SVC* - due to its ability to learn high-dimensional datasets such as this and handle non-linearly separable datasets with the help of kernels,

4. *RandomForest* - since it is an ensemble approach and can overcome the overfitting problem associated with Decision Trees,

5. *AdaBoost* - as this is a standard boosting algorithm which controls both the aspects of bias & variance, and is considered to be more effective, hence most classifiers predict a sample to belong to class 7, as shown in the confusion matrices.

6. *XGBoost*, an advanced implementation of gradient boosting algorithm,

7. *Neural Network using Keras* with loss as cross entropy and metrics as accuracy with a validation split

of 0.1, we plot epoch vs training loss and validation loss, epoch vs training accuracy and validation accuracy to check if it is overfitting, by the graphs we can easily see that after 10 epochs validation log loss starts increasing, and a few other methods. Hyperparameters are tuned for each model using grid search.
We have used simple parameters for RandomForest such 'n_estimators' and 'max_features', for SVC - 'C', 'decision_function_shape', for XGB - 'learning rate' and 'max_depth', and for ADABoost - 'learning_rate' and 'n_estimators'.

5. Results and Analysis

The accuracies, log-losses and confusion plots for each model is presented in the ipython notebooks (they need to be run on Jupyter for best visualization).

At first, we use a simple tf-idf transformer. With this pre-processing technique:

- The MultinomialNB classifier has very low cross-validation accuracies and high log-loss, as expected. (Log loss: 2.74217844806, Accuracy: 0.292984040952)
- The LogisticRegression model performs better, but is still prone to misclassification and over-classification for class 7. (Log loss: 1.76259870904, Accuracy: 0.34899126769)
- The SVM with Linear kernel performs slightly poorly than Logistic, but has better average precision and recall scores (Log loss: 1.95761040555, Accuracy: 0.322493224932, Avg Precision: 0.31, Avg Recall: 0.38)
- The SVM with rbf kernel performs extremely poorly, suggesting that this data cannot be separated with a gaussian kernel.
- The RandomForestClassifier performs the best in this setting, with Log loss: 1.72326575758 and Accuracy: 0.396567299006. It also has much better Avg Precision(0.34) and recall(0.40)

Hence, the RandomForestClassifier beats other techniques.
Next, we used a combination of tfidf and TruncatedSVD, and got slightly poorer results with a RandomForestClassifier(Log loss: 2.42271174732 and Accuracy: 0.314965371876). Hence we did not consider this pre-processing technique any further.
Afterwards, we used the Word2vec pre-processing technique and got the feature vectors in a reduced dimension space. We considered 2 stages of this - 1) Word2Vec trained on the training data, and 2) on both the training and test data. The reason we can do this is that Word2Vec is an unsupervised model, and simply takes the unlabeled text as input.
In the first stage, the performance of Logistic Regression increased a lot(Log loss: 1.65313998161 Accuracy: 0.417946401686).
For SVM under this setting, the performance increased as well(Log loss: 1.58455459641 and Accuracy: 0.426678711232), whereas RandomForest gives comparable results(Log loss: 2.02685740982 Accuracy: 0.43239987955) with slightly better precision and accuracy.
In the second stage, results did not improve much by adding the test data to the Word2Vec model.
Also, the attempts at using boosting techniques (AdaBoost, XGBoost, Gradient Boost Classifier) did not yield satisfactory results.

Hence, the best performance achieved till now has been that of Random Forest with the Word2Vec model. The performance of Doc2Vec was very similar to this. Using Neural Networks performed poorer compared to random forest with larger log loss. Upsampling did give better results when optimised and good validation score but with test set at Kaggle it performed poorly.

Accuracy=0.572932330827
Log loss 1.35031817006

	precision	recall	f1-score	support
avg / total	0.63	0.57	0.53	665

It shows that upsampling helped some of the classes with repeated sampling.
So with random forest and upsampling we got the best results for doc2vec for validation set but for Kaggle it was performing poorly.

Note: Please see the Ipython notebooks for more detailed results.

6. Contributions

6.1 Deliverables

Individual Deliverables-

Dhruva: ML Techniques, Feature extraction, External Data,Doc2Vec,Upscaling, Keras simple sequential Neural Network;

Aditya: NLP techniques, Feature extraction, Cross-validation, Performance evaluation, word2Vec, Boosting

Files: redef_cancer.ipynb, redef_cancer_predictions.ipynb : Aditya

Redef_cancer2.ipynb, Upsampling.ipynb : Dhruva

6.2 References

- Scikit Learn documentation

- http://linanqiu.github.io/2015/10/07/word2vec-sentiment/
- https://www.kaggle.com/c/word2vec-nlp-tutorial/discussion/12287
- https://www.kaggle.com/reiinakano/basic-nlp-bag-of-words-tf-idf-word2vec-lstm
- https://www.kaggle.com/the1owl/redefining-treatment-0-5745
- https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
- https://towardsdatascience.com/a-gentle-introduction-to-doc2vec-db3e8c0cce5e
- https://rare-technologies.com/word2vec-in-python-part-two-optimizing/
- https://rare-technologies.com/doc2vec-tutorial/
- http://scikit-learn.org/stable/modules/ensemble.htm
- https://elitedatascience.com/imbalanced-classes
- https://www.analyticsvidhya.com/blog/2017/03/imbalanced-classification-problem/#
- https://www.kaggle.com/alyosama/doc2vec-with-keras-0-77