Personalized Medicine: Redefining Cancer Treatment

Definition

Project Overview

A lot has been said during the past several years about how precision medicine, there have been a lot of research in this area, among them pathological diagnosis is still gold standard in clinical practice[2], in recent years, the development of genetic technology has brought major influence on the ways diseases like cancer are treated, for example, the science of “pharmacogenomics” was proposed, which identifies individuals who, based on their genotype information, will respond to a specific therapy [3]. But this is only partially happening due to the huge amount of manual work still required, while the collected data is increasing dramatically, the development of related analysis and prediction methods is still slow-moving, that is why NIPS is holding this competition and the problem this capstone project dedicates to resolve.

Problem Statement[1]

Once sequenced, 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).

Currently this interpretation of genetic mutations is being done manually. This is a very time-consuming task for the reason that a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

Based on that, the aim of this capstone project is to develop a Machine Learning algorithm that, using this knowledge base as a baseline, automatically classifies genetic variations.

Metrics

The performant will be evaluated by Multi Class Log Loss(MCLL) between the predicted probability and the observed target.
The metric can be described mathematically like this,

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} log(p_{i,j})$$

Where N is the number of observations, M is the number of class labels, $y_{i,j}$ is 1 if observation i is in class j and 0 otherwise, and $p_{i,j}$ is the predicted probability that observation i is in class j [5].

As the equation illustrates, MCLL quantifies the accuracy of a classifier by penalising false classifications, this is quite suitable to this problem, because medicine prediction is low false tolerant, especially for clinical decision. Another benefit of this metric is that it uses probability rather than boolean values, which makes the result even more accurate.

Analysis

Data Exploration

Datasets are provided via two different files, one(training/test_variants) provides information about the genetic mutations, whereas the other (training/test_text) provides the clinical evidence (text) that our human experts used to classify the genetic mutations. Both are linked via the ID field[5].

Variants exploration

Variants data

'training_variants.csv' is a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the aminoacid change for this mutations), Class (1-9 the class this genetic mutation has been classified on)

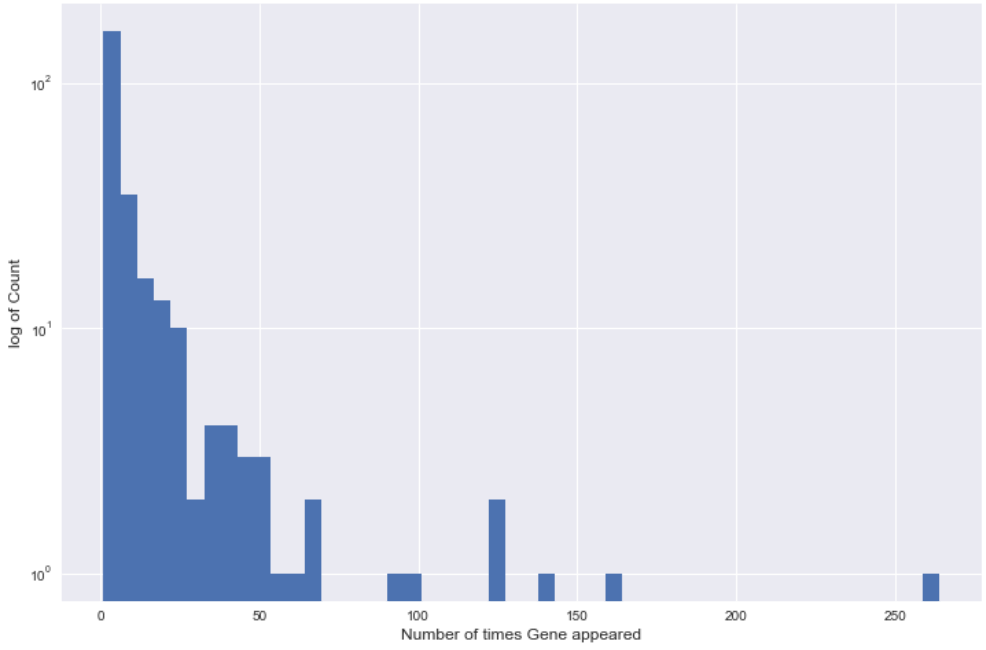
'test_variants.csv' is same as training data but with no Class field.

The head 5 of this file looks like this,

ID	Gene	Variation	Class
0	FAM58A	Truncating Mutations	1
1	CBL	W802*	2
2	CBL	Q249E	2
3	CBL	N454D	3
4	CBL	L399V	4

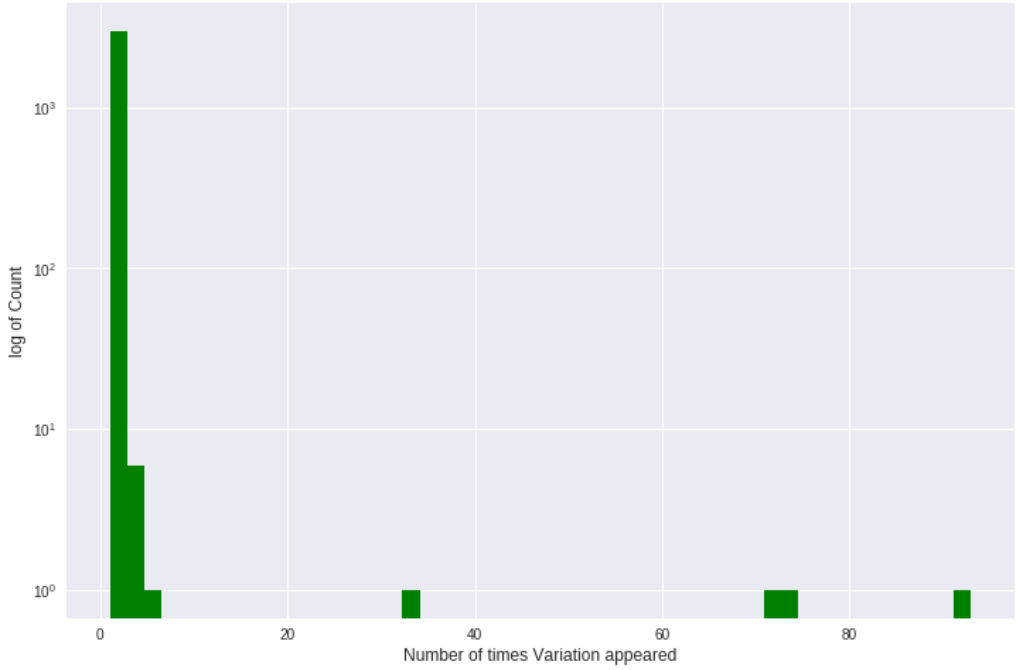
Let’s dig deeper about the feature fields

- The histogram of ‘Gene’ frequency



The x stands for the number of times gene appeared, it can be seen that the data is assembled mostly in the left part, which means that most genes appears less than 50 times, it also means that turning Gene into label directly may not be convenient enough for learning algorithms, as it may cause the problem of overfitting.

- The histogram of ‘Variants’ frequency



When it comes to the Variation data, the distribution is even more unbalanced, the estimated percentage of variation data which appears below the number of 5 is more than 95%. That means we should parse the variance text data into smaller characters which are more adequate to be used for training.

Text data

Traing_text.csv a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation). Test_text.csv is the same as training_text.csv in its structure.

First, we have a glance at what the text data looks like.

ID	Text	
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

It seems that the length of text is quite long, we can run this script to get its statistical describe based on its parsed words.

$$print\ train\_text\_df['Text'].apply(lambda\ x :\ len(x.split())).describe()$$

The result shows

count	mean	std	min	25%	50%	75%	max	dtype
3321.000000	9542.415537	7845.140815	1.000000	4733.000000	6871.000000	11996.000000	76708.000000	float64

The text are so long and could be challenging to interpret. Thus we may consider to rely on NLP tools which help to extract useful messages from the text, the processing of this clinical evidence might be a critical step for the approach of this problem.

Algorithms and Techniques

Based on the data we have, in order to address the problem, we should break through two obstacles. One is that we have to do some feature engineering of the text data, techniques in this area includes NLP tools such as Bag Of Words, TF-IDF, etc. The other is that we should pick up the most fittable Machine Learning methods and train the models with the best performance in prediction, during this approach , we will rely on the hottest 'XGboost' .

Bag Of Words

The first model we used is /Bag of Words/, which aims to represent features by counting the occurrence of each word, the corpus of document can be transferred into a matrix with one row per document and one column per word occurring in the corpus[6].

However, after testing in in the training datasets, we can find that features extracted by this method are too high in dimension. So the follow up step is to reduce dimension, there is a related package in Scikit-learn called 'decomposition' , it includes a lot of reduction techniques , among them TruncatedSVD is appropriate in this case , because it works efficiently with sparse matrices and it can transform transforms such matrices to a "semantic" space of low dimensionality[7].

TF-IDF

In the documents of clinical evidence, some words (e.g. "the", "is", "a") appears a lot but carries very little meaningful information about the actual contents of the documents. If the direct count data is feed directly to a classifier, those very frequent terms would shadow the frequencies of rarer yet more useful terms[6].

To address this problem, it is very common to use the tf-idf transform to re-weight the count features. Tf means term-frequency while tf–idf means term-frequency times inverse document-frequency. Scikit-learn also provides related class called "TfidfVectorizer" that combines all the options of vectorization and transformation in a single model.

Also, a reminder that the dimension reduction method is still needed , because TF-IDF only modifies the weights and has nothing to do with dimension.

Gradient boosting & XGBoost

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees[8]. In recent years, Gradient boosting has shined in many applications and data science competitions.

XGBoost is a open-source software library which provide gradient boosting framework, the implementation of the algorithm was engineered for computational speed and model performance. Actually, XGBoost has recently been dominating supervised machine learning especially in Kaggle competitions for structured or tabular data.

Focusing on our project, XGBoost can be applied after all the feature engineering work has been done, which means all the data has been formalized in new features previously.Belows are the params we need to pay more attention to[9].

- Booster[default = gbtree]: which booster to use, can be gblinear or dart.
- learning_rate[default = 0.3]: making the model more conservative by shrinking the weights on each step.
- max_depth[default = 6]: maximum depth of a tree, increase this value will make the model more complex / likely to be overfitting. 0 indicates no limit, limit is required for depth-wise grow policy.
- gamma [default=0, alias: min_split_loss] minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm will be. range: [0,∞]
- subsample [default=1]: subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.
- N_estimators: Number of boosted trees to fit.

Benchmark

The benchmark models we choose are Naive Bayes and Support Vector Machine(SVM), Naive Bayes is a simple classification technique, it assumes that the features are independent with each other in their values, the training process can be very efficient under the circumstance of supervised learning[10].
SVM is a widely used method in the problems of classification, the main target of SVM is to build a hyperlane that the distance from it to the nearest data point on each side is maximized, thus the data is separated in different categories[11].
We have noted that Sklearn has provided both the tools of Naive Bayes and SVM.

Methodology

Data Preprocessing

Genes and Variants

As the data exploration shows, the fields of variants file include genes and variants, both of them are text-based, in order to better train the learning model, we need to transform existing features and generalize more features when needed.

The first step we could start is to turn the text into labels, that will help us get a label matrix of the original data. Specifically, we have corresponding tool called *LabelEncoder*[12] in our toolbox *sklearn*, the tool helps normalize labels such that they contain only values between 0 and n_classes-1.

As we described in the histogram of genes and variants, the feature value is distributed in a wide range, over half the genes and variants appear only once, that would lead to the result that the labels produced by the first step are too many, that is not good for the efficiency of the training part. Thus it is necessary for us to break the short text into characters, which helps improve the predictive accuracy given the relevance of protein features[13]

Clinical Text

The average length of the text data sets is 9524 according to our statistics of text file above, so the preprocessing of the text file is a problem of NLP, the target of this procedure is to represent the long text with the most relevant features, that is the typical scenario where TF-IDF[14] applied in.

TF-IDF is short for term frequency-inverse document frequency, it is widely used in information retrieval and recommender system, the main idea is to reflect how important a word is to a document in a collection. Given the text file we have, we will get a matrix of important words in each text, which feeds the model training better.

However, after running the algorithm of TF-IDF, the output matrix is much higher in dimension than expected, the following-up step is how to reduce the dimension while preserving the similarity structure among features. The related tool(*TrunctedSVD*) we applied comes from the package of sklearn, this tool has a benefit that it works well with sparse matrix.

Related code looks like this

```
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.decomposition import TruncatedSVD

def text_tfidf(df):
      count_vectorizer = TfidfVectorizer(analyzer="word",
tokenizer=nltk.word_tokenize,
                                         preprocessor=None,
stop_words='english', max_features=None)
      tfidf = count_vectorizer.fit_transform(df['Text'])

      svd = TruncatedSVD(n_components=50, n_iter=25, random_state=12)
      truncated_tfidf = svd.fit_transform(tfidf)
      return truncated_tfidf
```

Model Training

During this stage, the predictive model was trained on the preprocessed training data. As we illustrated above, the main staff of this stage is basically parameter-tuning, the methodology we choose is *GridSearch*[15], which is an exhaustive searching of manually specified subset of parameters. To evaluate the performance of grid search and reduce the risk of overfitting, we choose Cross Validation (CV) as the training framework and got the predict result after 5 rounds training.

Regarding the selection of parameter subset, we tuned the most related ones, including “max_depth”, “learning_rate”, “gamma” and “n_estimators”

Related codes are showed below

```
def xgb_cv_fit(train_X, train_y):

      cls = xgb.XGBClassifier(objective='multi:softprob', subsample=0.5,
colsample_bytree=0.5, random_state=0)

      param_grid = {'max_depth': [3, 4, 5, 6], 'learning_rate': [0.05,
0.1, 0.3],'n_estimators': [25, 50, 100]}

      model = grid_search.GridSearchCV(estimator=cls,
param_grid=param_grid, scoring='accuracy', verbose=10,n_jobs=1,
iid=True, refit=True, cv=5)

      model.fit(train_X, train_y)

      probas = model.predict_proba(train_X)

      print("Best score: %0.3f" % model.best_score_)
```

```
    print("Best parameters set:")

    best_parameters = model.best_estimator_.get_params()

    for param_name in sorted(param_grid.keys()):

    print("\t%s: %r" % (param_name, best_parameters[param_name]))

    print('Log loss: {}'.format(log_loss(train_y, probas)))
```

At first I wrote the scripts of cross validation myself until I found that sklearn offers related package along with the function of gridsearch, and things became easier, the obstacle lies in the data preparation and efficiency, as we illustrated before, we did a lot of preprocessing on genes data and clinical text data, however, when we combined them together, the data sets from differenct parts are not same formated, some of them are dataframe(Pandas), the others are mere arrays, finally we concated them in the format of numpy array.
The other problem is efficiency, the time cost mainly lies in the dimensionality reduction and cross validation, after several rounds of justification, we choose 50 as the components of TrunctedSVD and 5 rounds of cross validation and got a better balance of accuracy and efficiency.

As to the training of benchmark models, we build a function to reduce the repetitive work of cross validation and fitting:

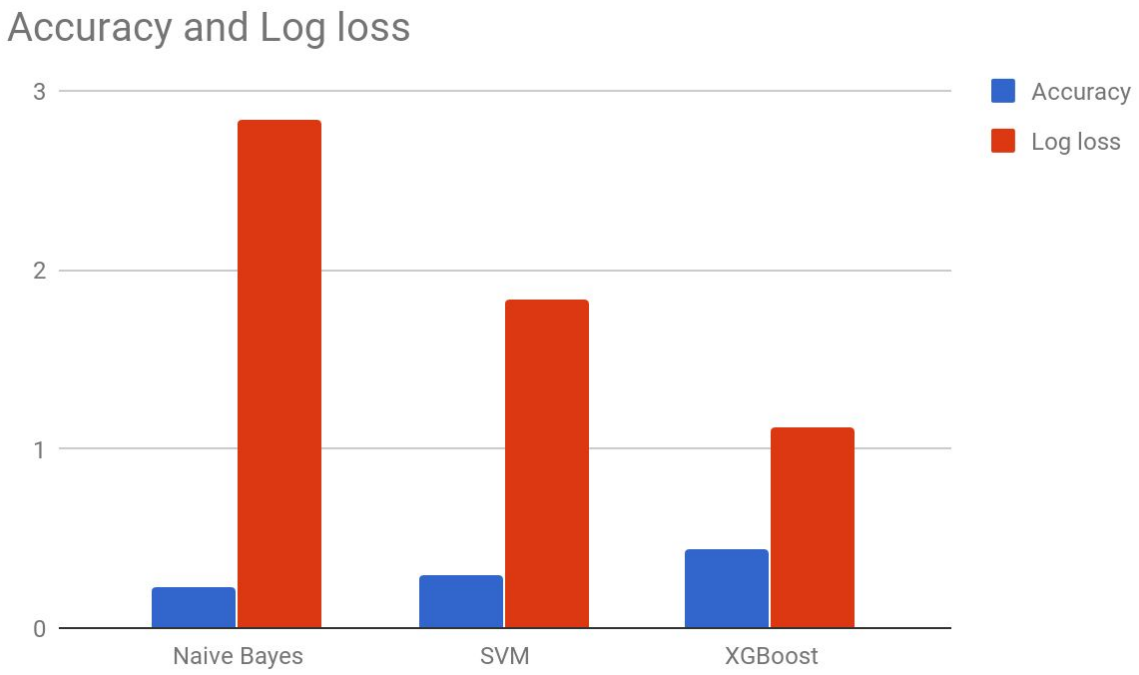
```
def benchmark_fit(clf, train_x, train_y):

  probas = cross_val_predict(clf, train_x, train_y, cv=StratifiedKFold(random_state=0),
                             n_jobs=-1, method='predict_proba', verbose=2)
  pred_indices = np.argmax(probas, axis=1)
  classes = np.unique(train_y)
  preds = classes[pred_indices]
  print('Accuracy: {}'.format(accuracy_score(train_y, preds)))
  print('Log loss: {}'.format(log_loss(train_y, probas)))
```

Results

The result shows that XGBoost's best accuracy score is 0.437, which is under the chosen best parameters of (gamma:0, learning_rate:0.1, max_depth: 3, n_estimators:25)
The log loss value between predictions and true targets is 1.116. As the table and bars below show, compared to the other two benchmark models, XGBoost is higher in accuracy and lower in the value of Log loss.

	Naive Bayes	SVM	XGBoost
Accuracy	0.232	0.293	0.437
Log loss	2.833	1.830	1.116



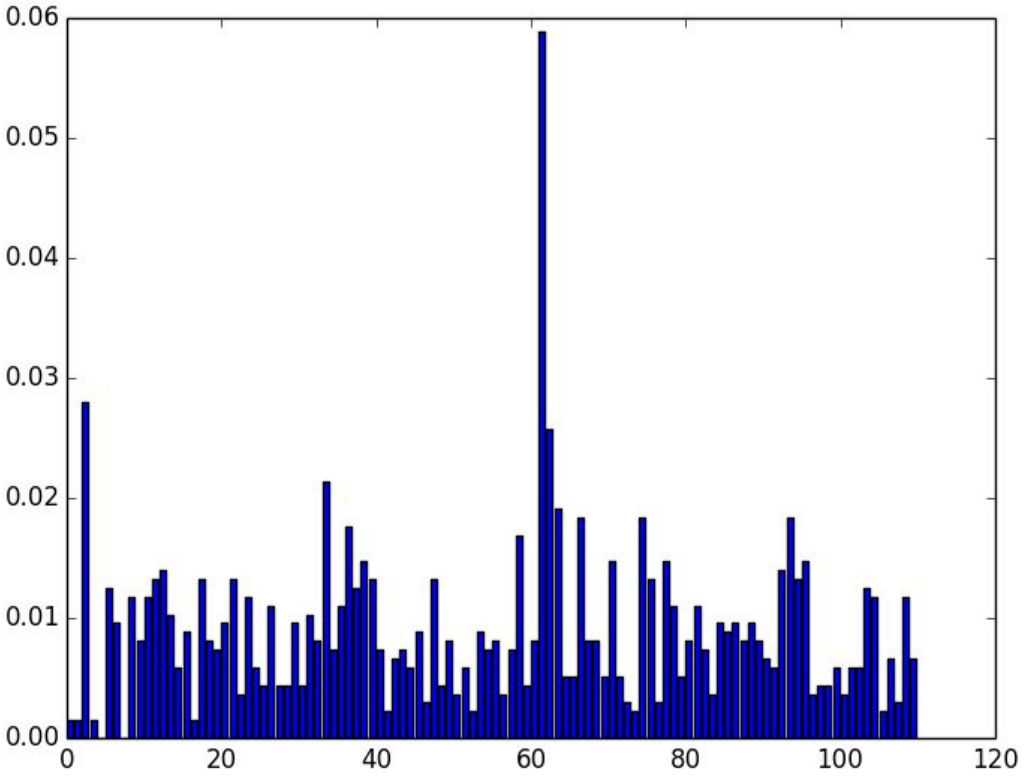
Due to the fact that we trained the model through cross validation, we believe that this result is robust enough in the scope of given training data, as to the unknown data, the model may need further justification especially in the selection of parameters.

Conclusion

This project attracts me because of its potential application in real world, as we know that cancer is a major killer of human beings, the treatment research in this area has last for almost a century, however, there is no definite solution to tackle this disease. The most reliable therapies in practice relied on previous experience, which basically stay in experts' minds. The current situation is starting to be promoted for the prevalence of Data Science, with the help of Machine Learning we expect to discover more useful information from mountains of medical data.

Regarding the details of this project, several techniques were tried to make the most of the given data. At first, we preprocessed the raw data of short words and long text. By symbolizing short words into labels and parsing them into characters, then we met the problem of high dimensionality and used TrunctedSVD to perform dimensionality reduction.

When it comes to the long clinical text, we mainly relied on TF-IDF to find the most important significant words in a text message.
Then the training part is basically the tuning of XGBoost, we applied Cross-validation and grid search to ensure that the parameters we use are mostly optimized.According to the model result, the feature importance can be depicted as below:



Where x stands for the index of features, y stands for the important rates, as we can see, most features have similar rates, except No.62 which comes from the results of tfidf processed text.

This project can be much more complicated than it seems to be. As we dig more for a better result, at least two aspects we can explore further, the first one is the medical domain knowledge, e.g. how would the meaning of genes variation helps us better symbolizing the given data so we can achieve better prediction, the second potential breaking through is the application of Natual Language Processing(NLP), especially with the help of deep learning, that will make the clinical text more valuable.

Reference

[1] https://www.kaggle.com/c/msk-redefining-cancer-treatment
[2] Verma M. Personalized Medicine and Cancer. *Journal of Personalized Medicine*. 2012;2(1):1-14. doi:10.3390/jpm2010001.

[3]Schroth W., Goetz M.P., Hamann U., Fasching P.A., Schmidt M., Winter S., Fritz P., Simon W., Suman V.J., Ames M.M., Safgren S.L., Kuffel M.J., Ulmer H.U., Boländer J., Strick R., Beckmann M.W., Koelbl H, Weinshilboum R.M., Ingle J.N., Eichelbaum M., Schwab M., Brauch H. Association between CYP2D6 polymorphisms and outcomes among women with early stage breast cancer treated with tamoxifen. JAMA. 2009;302:1429–1436. doi: 10.1001/jama.2009.1420.
[4] https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
[5] https://www.kaggle.com/wiki/MultiClassLogLoss
[6] http://scikit-learn.org/stable/modules/feature_extraction.html
[7] http://scikit-learn.org/stable/modules/decomposition.html#lsa
[8] https://xgboost.readthedocs.io/en/latest/get_started/
https://en.wikipedia.org/wiki/Gradient_boosting
[9]http://xgboost.readthedocs.io/en/latest/parameter.html#general-parameters
[10]https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[11]https://en.wikipedia.org/wiki/Support_vector_machine
[12] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html
[13]https://www.kaggle.com/danofer/genetic-variants-to-protein-features
[14]https://en.wikipedia.org/wiki/Tf%E2%80%93idf
[15]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html