COMPSCI4061 Machine Learning H Coursework Report

- The Regression Part
- The Classification Part

The Regression Part

Firstly, let us look at linear regression. Since we have training data and testing data, the linear regression is a supervised machine learning problem. The idea of regression is to find a model that could fit the training data sets with the least loss.

Question 1

What are the models behind the two regression algorithms you chose and implemented?

Bayesian Linear Regression

Bayesian linear regression is a regression approach that takes statistical analysis under the context of Bayesian inference. Its errors follow the normal distribution. We assume a particular form of the prior distribution and get the results from the posterior probability distributions.

According to the Bayes rule

$$p(\mathbf{w}|\mathbf{X},\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{X},\mathbf{w})p(\mathbf{w})}{p(\mathbf{t}|\mathbf{X})}$$

Posterior density equals the likelihood of multiple prior density divide by marginal likelihood.

$$\boldsymbol{\Sigma} = \left(\frac{1}{\sigma^2}\mathbf{X}^\mathsf{T}\mathbf{X} + \mathbf{S}^{-1}\right)^{-1} \qquad \boldsymbol{\mu} = \frac{1}{\sigma^2}\boldsymbol{\Sigma}\mathbf{X}^\mathsf{T}\mathbf{t}$$

But computing the posterior is hard since the marginal likelihood is hard to compute. Only in the case when the prior and likelihood are conjugate, we know the form of posterior, therefore we know the form of the normalizing constant. So, we do not need the marginal likelihood. By calculating the covariance and mean of the training dataset, we can simply find its covariance and mean.

By minimizing loss with regularization, we adapt multivariate normal with post mean and post covariance to get result w. Finally, multiple the test data set with w to get the test result.

Maximum Likelihood Regression

Maximum likelihood Estimation of Linear Regression is a probabilistic framework for efficiently getting the probability distribution and finding the best model to describe a dataset. Specifically, the model choosing, and parameters define as a modeling hypothesis. We are trying to find the modeling hypothesis that maximizes the likelihood function

Here are functions for maximum likelihood estimator for w and sigma square

$$\hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t}, \;\; \hat{\sigma^2} = \frac{1}{N}\left(\mathbf{t} - \mathbf{X}\hat{\mathbf{w}}\right)^T\left(\mathbf{t} - \mathbf{X}\hat{\mathbf{w}}\right)$$

Likelihood evaluates the quantity obtained when evaluating the density. The higher the value, the more likely training data is fitted. For each input-response pair, we have a Gaussian likelihood to compute the log-joint likelihood at the maximum likelihood estimates. Assume that the training data is independent, we combine them to get the joint likelihood.

$$\underset{\mathbf{w},\sigma^2}{\operatorname{argmax}} \prod_{n=1}^{N} p(t_n|\mathbf{w}, \mathbf{x}_n, \sigma^2)$$

Then, we calculate the covariance matrix of w

$$cov\{\mathbf{w}\} = \hat{\sigma^2}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$$

Finally, we get sample w by multivariate normal between the calculated w and cov{w}. Multiple the test data set with sample w to get the test result.

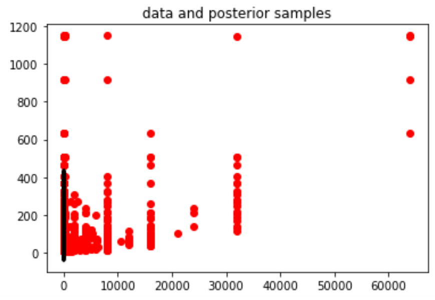
Question 2

What is your experiment setup for training these methods?

Bayesian Linear Regression

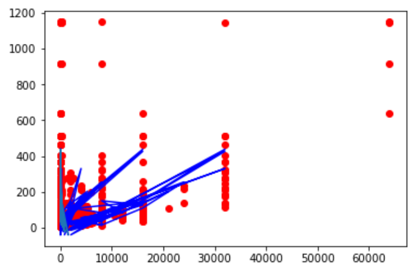
1. Split the dataset into six parts by its features ---- Cache memory size, the minimum and the maximum number of I/O channels, machine cycle time, and minimum and maximum main memory. Each of them is a (168,1) list.
2. Define a Gaussian prior over $\mathbf{w}$, with mean 0, a (2,2) covariance matrix, and the fixed $\sigma 2=2$.

3. Using NumPy's multivariate normal to generate samples from a multivariate Gaussian and sampling some $\mathbf{w}$ vectors and plotting the models with all features.



4. Write functions to construct polynomial design matrix, and compute posterior mean and covariance where $\mathbf{S}$ is the covariance matrix of the prior $p(\mathbf{w})$. Since there are 6 features, we add each polynomial dataset and compute the expected result sequentially.

5. Look at functions for posterior prediction by plotting the error bar for general checking.



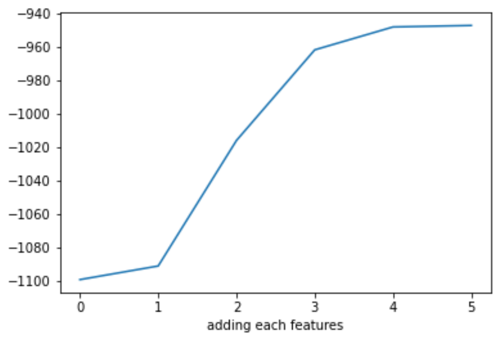
6. when the error bar looks fine, use w to generate test results from the test dataset.

Maximum Likelihood Regression

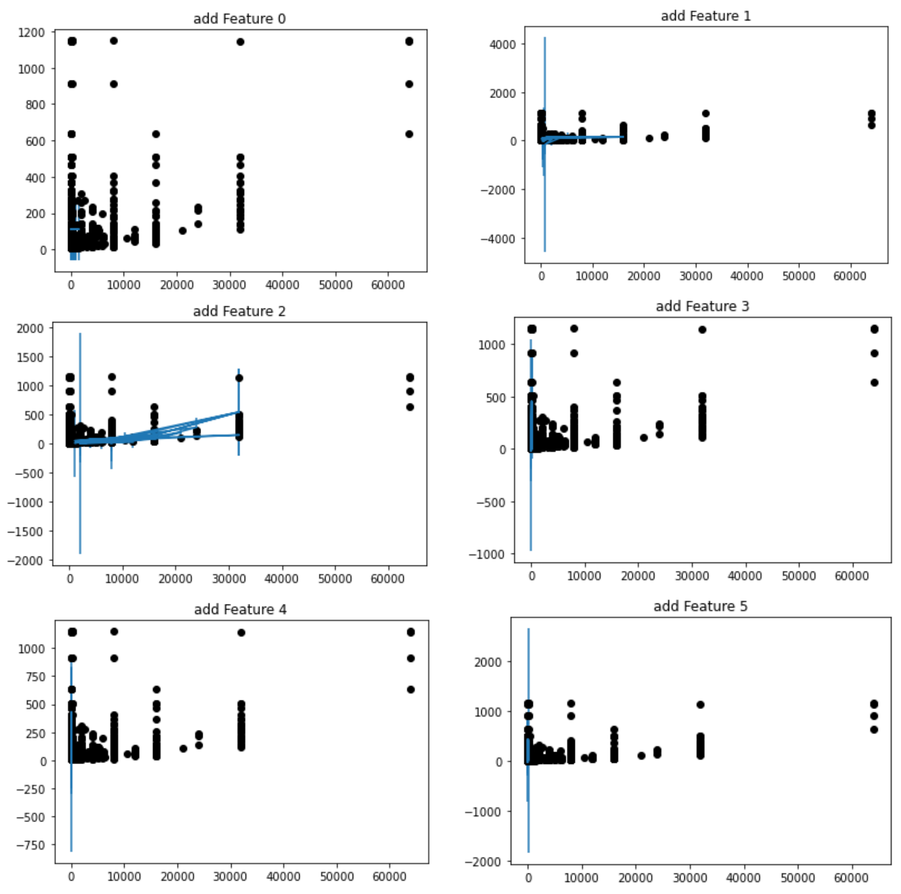
1. Split the dataset into six parts by its features ---- Cache memory size, the minimum and the maximum number of I/O channels, machine cycle time, and minimum and maximum main memory. Each of them is a (168,1) list.

2. Write a function to compute the log of gaussian pdf, maximum likelihood estimate of w, maximum likelihood estimate of sigma square, and construct the polynomial design matrix

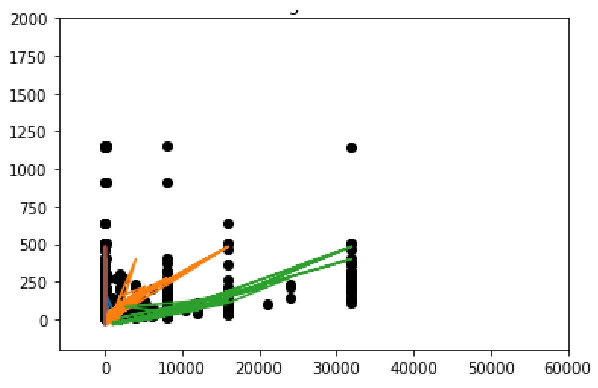
3. Test the joint likelihood against the adding of features to check the influence of each feature. Loop over all features, use the maximum likelihood estimate of w and maximum likelihood estimate of sigma square to compute the log-joint likelihood at the maximum likelihood estimates. The plot lost between adding features.



4. Predictive variance example, step up



5. Look at functions for posterior prediction by plotting the error bar for general checking.



6.When the error bar looks fine, use w to generate test results from the test dataset.

How are you going to measure the performance of your regression algorithms?

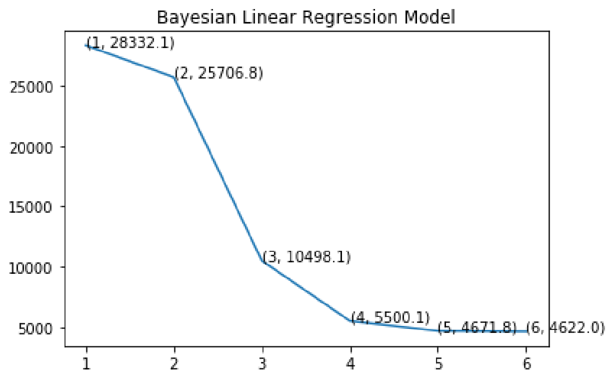
$$L(\mathbf{w}) = \frac{1}{N}(\mathbf{t}-\mathbf{X}\mathbf{w})^T(\mathbf{t}-\mathbf{X}\mathbf{w})$$

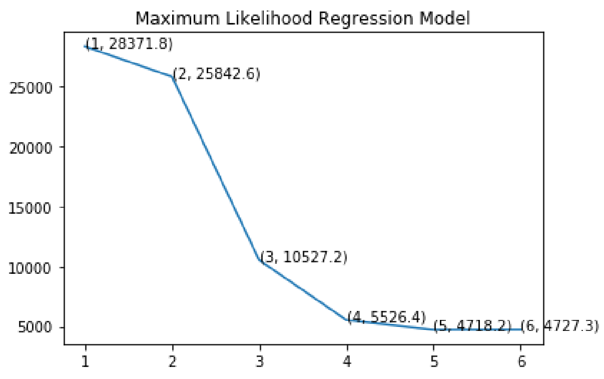
Plotting the predicted values against the real value is the best way to measure the goodness of a regression model. I use average squared loss, which measures the average error performed by the model in predicting the outcome for an observation. Mathematically, it is the average squared difference between the observed actual outcome values and the values predicted by the model. The lower the average squared loss reaches, the better the model accuracy will get.
Alternatively, we can test our test dataset on the Kaggle prediction. Somehow, it performs as a cross-validation way to check the general performance. However, it is not as explicit as the average squared loss.

Question 3

Compare the performance of these two algorithms.

In this section, I will adapt the average squared loss measurement onto the Bayesian Linear Regression Model and the Maximum Likelihood Regression Model.
In order to compare those two models in detail, I choose to construct polynomials on training data. For each constructed feature, I generate their model and use the original data in the generated model to calculate predict data. Finally, computing the loss between the actual train data and test data by average squared loss. The least result indicates the better performance this model may have.





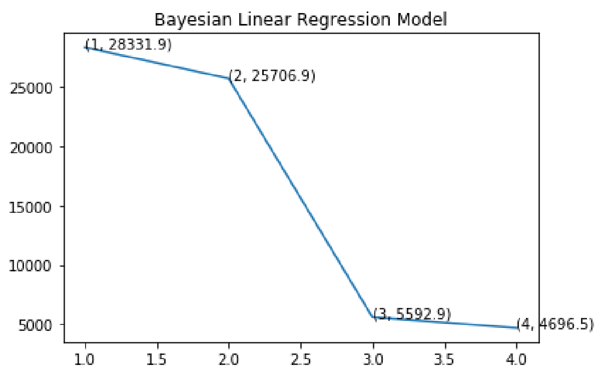
As we can see in the plot, in general, adding features may decrease the model loss. For some reason, in the Maximum Likelihood Regression Model, the performance of 5 features is better than adding one more feature. The performance of the Bayesian Linear Regression Model is slightly better than the Maximum Likelihood Regression Model.
Bayesian Linear Regression Model fits the training data well and Maximum Likelihood Regression Model could perform a better prediction on a wide scope of the dataset. Thus, the Maximum Likelihood Regression Model has a likely equal performance on general practical datasets comparing with Bayesian Linear Regression Model.

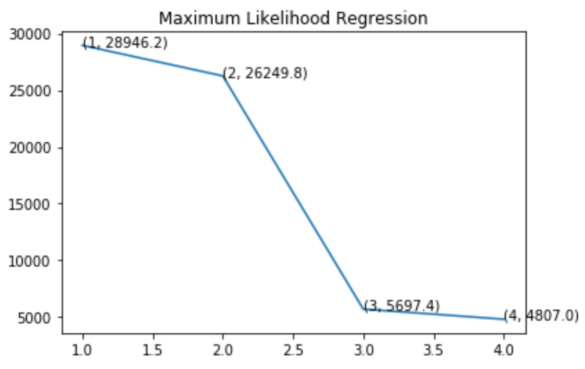
Question 4

Can you obtain better performance by using only a subset of the features?

For both models, I tried to combine some “sharpen” features. I combined the minimum and the maximum main memory as the main memory feature and the minimum and the maximum number of I/O channels as the I/O channels feature. So, I subtracted the feature number from 6 to 4.

After that, I adapted the model based on my model setup. And I used the polynomial type of the original combined dataset to predict the test dataset. Finally, I calculated the average squared loss and compared the performance between the flatten feature prediction.





According to the average squared loss score, the combining feature penalizes the Maximum Likelihood Regression Model but slightly improves the Maximum Likelihood Regression Model. And in the Kaggle competition testing data score, there is not a better performance shown.

Thus, in our hypothesis to combine the minimum and maximum dataset features, we cannot improve model performance. However, there may exist other combination ways to eliminate dataset noise.

The Classification Part

We will then talk about the part of classification; since we have got training data and the training classes(labels), we are dealing with the supervised machine learning problem. The idea of classification is to find the Model that could fit and learn from the training data sets reasonably to make predictions on the classes or the incoming class-unidentified data labels.

1.This section will answer the models behind the two classification algorithms we chose and implemented

1.1. First Model

The first Model implemented is the K-Nearest-Neighbors classifier, a non-probabilistic classifier, and is generally used for prediction and classification for discrete-valued classes. The algorithm's basic idea is to find the K closest training points to each of the testing points and assume the class of that single testing point is the class of the majority training points within those k points around the testing point. The algorithm is implemented in CM-k-Nearest-Neighbors. file and called knn_classifier.

Then, we will look into the implementation procedure to explain the mathematical details of this Model. First of all, the function will take four parameters, the trainX(training data x with features), the traint(training class), the test_data(single row of the testing data x with features), and the K value(a constant). Initially, we calculate the distance between the test_data x, which is one row of the whole testing data. The entire training data x uses Euclidean distance, which is one of the most frequently used distance metric; we use this approach because it could generally provide accurate distance between 2 data, and it is easy to use. The mathematical formula is represented as:

$$Euclidean\ Distance = d = \sqrt{\sum_{i=1}^{N} \left(Xi - Yi \right)^2}$$

Thus, we could get the distance or the similarity of each training data x to the testing data x by doing subtraction on their features using:

distances = np.sqrt(((trainX - test_data)2).sum(axis=1))**
so that we can observe the similarity of two different data set based on their feature differences, which means that if their feature values are pretty similar, it is highly likely that they are coming from the same classes, then we sort the distances in ascending order to obtain the first K distance of training data x, after that, we count the training data x for each class and assign the greater counted class to the specific testing data x as well as calculating the predicted score for that testing data x. Finally, we combine the predicted class and predicted score into a dictionary and return them.

1.2. Second Model

The second Model has been applied is the Naïve Bayes classifier, which is a probabilistic based classifier relying on Bayes's theorem, and it is commonly utilized in text classification and grouping problem with multiple classes. The idea of this metric is that the class of the data points is determined based on the value of posterior probability where the posterior, which is the probability that x belongs to t=k, is calculated by the normalized multiplication of the likelihood that x is sampled from t=k and the prior (The probability of the occurrence of the class k: in all of the classes):

$$P(t_{\text{new}} = k | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) = \frac{p(\mathbf{x}_{\text{new}} | t_{\text{new}} = k, \mathbf{X}, \mathbf{t}) p(t_{\text{new}} = k)}{\sum_j p(\mathbf{x}_{\text{new}} | t_{\text{new}} = j, \mathbf{X}, \mathbf{t}) P(t_{\text{new}} = j)}$$

Furthermore, the likelihood probability for calculating posterior probability is calculated using Gaussian Distribution function (here we assume the data is usually distributed) denoted as:

$$\mathcal{N}(x\ ;\ \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2}(x-\mu)^2/\sigma^2\right]$$

Now, the implementation will be explained to observe mathematic details. First of all, we calculate the mean μ and variance σ^2 (applying on features) and prior of training data for each class and calculate the likelihood probability using gaussian distribution provided above with mean and variance as well as testing data x (the higher the distance of testing data x to training data x 's mean, the lower the likelihood probability(testing data x is not likely from the class of training data x)). afterward, the prior is multiplied, and the whole computation is normalized to attain the posterior probability. Finally, the training data class of the higher posterior probability is selected as the class for testing data x since the testing data x has a lower distance to that training data x 's mean, and the normalized probability itself is assigned as the predicted score.

1.3. Benchmark for Two Models

The differences that make the Naïve Bayes approach different from the K-Nearest-Neighbors method are:

1. It assumes that a specific feature in the class is independent of other features within that class, so it could separate data using features based on this assumption. Whereas KNN separates the data based on each pair data's feature similarity regardless of the differences between distinct features, Naïve Bayes assumes that the data is normally distributed.
2. Naïve Bayes classifier is linear classifier whereas KNN is contrary, it could process quickly on big data.

2.This section will demonstrate the experiment setup for training the classifiers and performance measurements

2.1. For K-Nearest-Neighbors

Firstly, For the experiment setup stage, the training and testing data are loaded using NumPy, then, the VarianceThreshold with the threshold set as 0 is utilized to filter out the constant features for the training x and testing x data as the constant features do not help in classification. After eliminating the constant features, we apply StandardScaler to normalize the training and testing x data. The reason why we normalize the data is that we want the independent features to be in the same range, and that is essential for distance-based models.

Then, moving to the training and performance measuring stage, before moving to Repeated k-fold cross-validation, we need to introduce the k-fold cross-validation, which is the process of separating the training data into k-folds and take N-1 as training data and 1 as testing data for validation to each fold, subsequently, The Repeated k-Fold cross-validation technique with 10 times repetition and 10-Fold separation is used to cross-validate on the data to measure the generalized performance and obtain the best k for KNN classification. The idea of Repeated K-Fold cross-validation is to repeat certain times to test the accuracy (which is calculated by 0/1 loss)of k-fold cross-validation and measure the mean result across all folds to improve the approximate performance of the Model by reducing bias. In this case, if we cannot get a precise outcome of the Model from k-fold cross-validation, we will get a relatively accurate performance of the Model by repeating it several times. Thus, by performing Repeated k-fold cross-validation on all features' training data, the best k, which is **9**, is found from the training procedure to ensure the average accuracy of this specific cross-validation to **91.05%.** After executing this performance measure, a sequence of other performance measurement approaches is also applied to confirm the performance of the Model. They are:

(1).0/1 loss based cross-validation with fixed random state, ROC (receiver operating curve), is utilized to represent the trade-off between experimental sensitivity and specificity.
(2). AUC (Area under the curve) which will display how correctly the positive rate (recall) and false-positive rate trade-off,
(3). confusion matrix (present number of TP, TN, FP, FN of classification),
(4). Precision, which refers to the result's proportion, recall gauges the fraction of overall relevant results accurately classified by Model, and f1-score indicates the weighted average of the precision and recall.

Primarily, we perform cross validation to split training data and testing data into 7:3 ,then By applying Best K=9 to the model and fit the training data to predict on test data, consequently, we have got the Accuracy of 86.7% , AUC=0.94, precision: class=1.0 (epithelial) = 0.926, precision : class = 2.0 (stromal) = 0.818 , recall : class = 1.0 (epithelial) = 0.806, recall : class = 2.0 (stromal) = 0.913 and f1-score : class = 1.0 (epithelial) = 0.862, f1-score: class=2.0

(stromal)=0.871, The ROC (left) and confusion matrix(right), those are the good quality values from prediction, and it indicates that our model is trained decently.

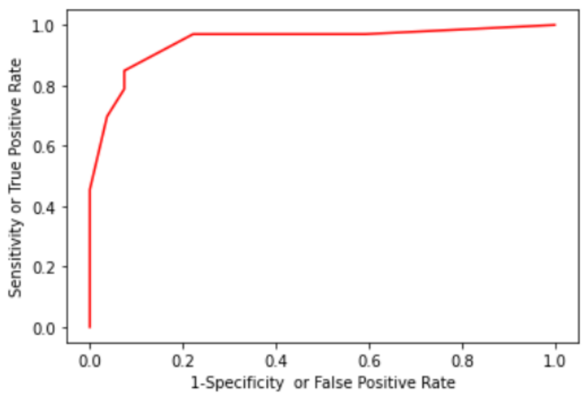


Figure 1: ROC for KNN with all features

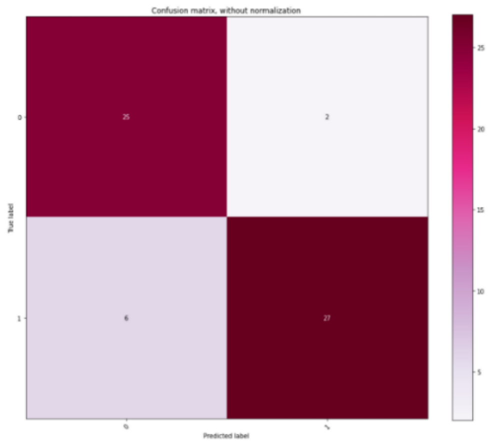


Figure 2: confusion matrix for KNN with all features

2.2.For Naïve Bayes

As the same experiment setup procedure as KNN for comparison intention, we first remove the constant features and normalize the data. then, for the training and performance measuring stage, we did Repeated k-fold cross-validation with fixed randomness on training data with all features again to test the generalizability and overall accuracy of the prediction of the Naïve Bayes classifier, subsequently, the result has been collected ,the average accuracy of the repeated k-fold cross-validation for this Model is **83.1%**, afterwards, the same confirmation performance metrics that have been applied to measure the KNN model have been applied to Naïve Bayes classifier as well, which gives us **85%** (which is generally similar to the value produced by Repeated Cross Validation) accuracy for cross-validation of fixed randomness with training and testing data divided with ratio 7:3, AUC of **0.9023569023569024,**

Precision: class = 1.0 (epithelial) = **0.926**, precision : class = 2.0 (stromal) = **0.788** , recall : class = 1.0 (epithelial) = **0.781**, recall : class = 2.0 (stromal) = **0.929** and f1-score : class=1.0 (epithelial) = **0.847**, f1-score: class= 2.0 (stromal) = **0.852**, The ROC (left) and confusion matrix(right),

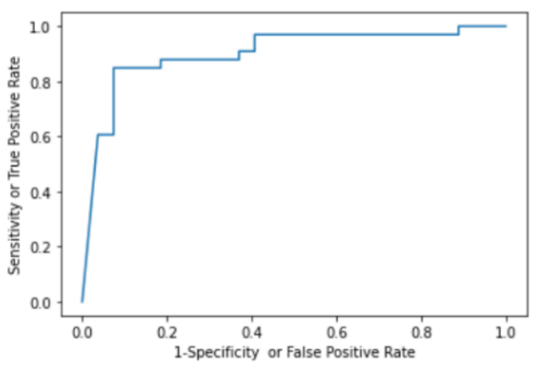


Figure 3: ROC for Naïve Bayes with all features

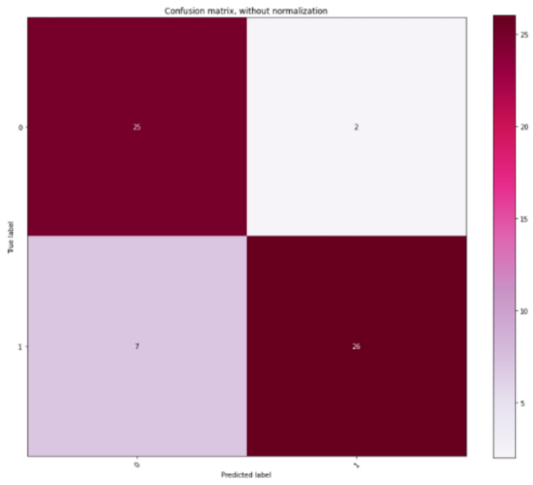


Figure 4: confusion matrix for Naïve Bayes with all features

3.Compare the performance of these two algorithms

Before comparison, the metrics utilized to help with comparison will be introduced; they are accuracy validated by cross-validation with data separated in ratio 7:3, ROC graph, AUC, confusion matrix, Precision, Recall, and f1-score measurements.
Then, we introduce some of the properties of those classification measuring methods:
(1). The precision measures the correctness of the positive identification for Model, higher the better
(1). The recall measures the correctness of actual positive identification; the higher, the better.
(3). The confusion matrix provides the number of TP, TF, FP, FN of the Model.
(2). F1-score is used more than accuracy when the TN and TP have relatively different amounts.
(3). As this is Binary classification, the AUC is used to measure the classifier's performance; the higher result represents a better Model.
By observing the performance measurements listed above, we find that the overall performance of KNN is better than that of the Naïve Bayes classifier as it generally provides higher accuracy, AUC, precision, recall, and f1-score; the higher AUC means a better trade-off between recall of FP rate, the higher precision indicates a higher relevance of the result, the higher recall return, the higher entire relevant of results precisely classified by the classifier, the higher f1-score indicate the relative higher harmonic average of the Model's precision and recall. KNN also obtain lesser FN by observing the confusion matrix above. Moreover, the ROC of the KNN offers a better trade-off between clinical sensitivity and specificity than the Naïve bays model, as we can see from the graph below.

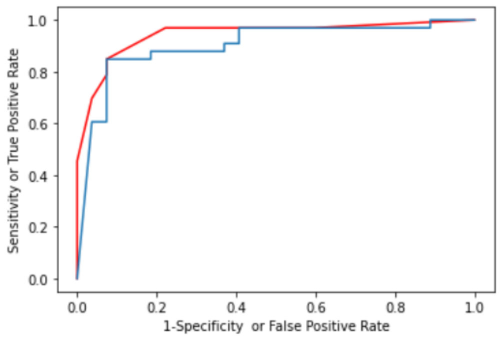


Figure 5: ROC comparison for Naïve Bayes(blue) and KNN (red) with all features

However, let us look into the results. We will find that the recall for label stromal=0.929 of Naïve Bayes classifier is slightly higher than that of KNN classifier, which is 0.912, which gives us the insights that Naïve Bayes classifier may find a more corrected positive instance of stromal than KNN would do. If we are looking for a correct number of normal cells, Naïve Bayes might be more useful than the KNN model.
In conclusion, the overall performance of the KNN classifier is better than that of the Naïve Bayes classifier when we deploy all the 112 features into classification. The reason could be that the Naïve Bayes classifier relies more on the independent property of the features to determine the labels for the data. In contrast, KNN relies on the similarity of the features to determine the labels. So if there are correlated features involved in the

calculation, the result would not be optimistic; so, the solution would be to deploy a feature selection method to find a subset of features to obtain the better performance for the Naïve Bayes classifier than apply all of the features.

4, This section will confirm that if we use feature selection approaches to obtain a subset of features, we will make better performance of those two classifiers

Many feature selection methods exist; to perform the correct feature selection method, we need first to acknowledge what kind of learning we are performing (supervised or unsupervised). The selection would be demonstrated as the figure below:

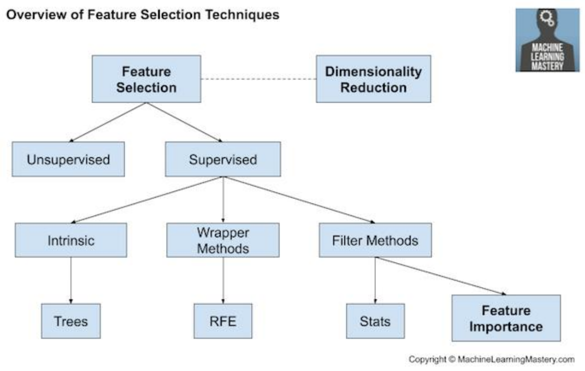


Figure 6: Overview of Feature Selection Techniques (Jason 2020)

All our classification model is supervised; thus, we could use Intrinsic, Wrapper Methods, and Filter Methods. Our idea is to construct a pipeline of feature extraction, which involves the first stage of Filtering methods processing and the second stage of Wrapping method processing. For the filter methods, the type of the Model's input and output data will take part in choosing the correlation statics for the method. The specific choosing strategy is shown in the figure below:

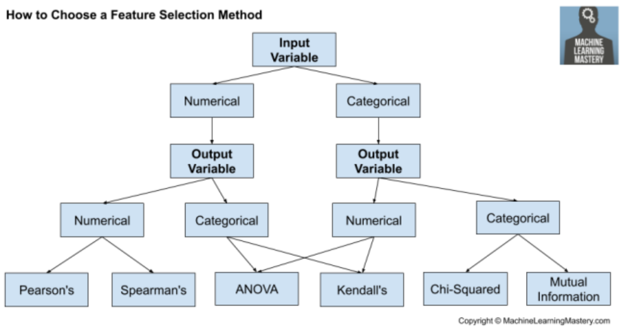


Figure 7: How to Choose Feature Selection Methods for Machine Learning (Jason 2020)

Since the input data of our Model is Numerical and the output of our Model, which is binary labels for cell type, is categorical, we choose ANOVA as our statistic model; we can apply it along with the sklearn library SelectBest(Select best k features) by importing f_classif() from sklearn library.

For the Wrapper method, we will utilize RFE (Recursive Feature Elimination), which will examine a subset of features by removing useless features from the Model by fitting the Model several times. The determination of whether to keep the feature will be applied using the feature importance attribute of the Model.

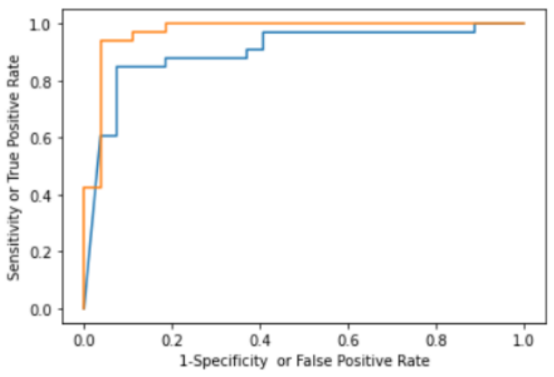
4.1. For Naïve Bayes model

We initially choose 50 features using SelectBest method and apply Wrapping method of RFE to recursively eliminate the features until the last 20 top feature remains. Afterward, the same process illustrated above in section 2, which utilizes Repeated k-fold cross-validation, is again performed to test and validate our model using training data. After several rounds of training and choosing a different number of top features, we find out that the top 10 of the features

(which are'SkewnessLayer1', 'RatioLayer3', 'HSITransformationHueRLayer1GLayer2BLayer3','Areaofsubobjectsmean1Pxl', 'Areaofsubobjectsstddev1Pxl','GLCMHomogeneityquick811Layer1alldir', 'GLCMContrastquick811Layer1alldir','GLCMMeanquick811Layer1alldir', 'AreaofsubobjectsNucleus1Pxl','ContrasttoneighborpixelsLayer33')

bring the relative higher generalized accuracy (**90.15% (112 features 83.1%)**) than other attempts. Then, to confirm the prediction accuracy of our model, we applied the same sequence of performance measurement methods illustrated above to help with validation. Firstly, the 0/1 loss based cross-validation which partition training data into 7:3 is again performed, and as the outcome, we have accomplished the accuracy of **90% (112 features 85%)** (which is generally similar to the value produced by Repeated Cross Validation), AUC of **0.9719 (112 features 0.9023)**,

precision: class=1.0(epithelial)=**0.963(112 features 0.926,)**,
precision: class=2.0(stromal)=**0.848**, **(112 features 0.788)**
recall: class=1.0(epithelial)=**0.839(112 features 0.781)**,
recall: class=2.0(stromal)=**0.966 (112 features 0.929)** and
f1-score:class=1.0(epithelial)=**0.897(112 features 0.847)**
f1-score: class=2.0(stromal)=**0.903(112 features 0.852)**,
The ROC comparison of all features and subset features(left) and confusion matrix (right):



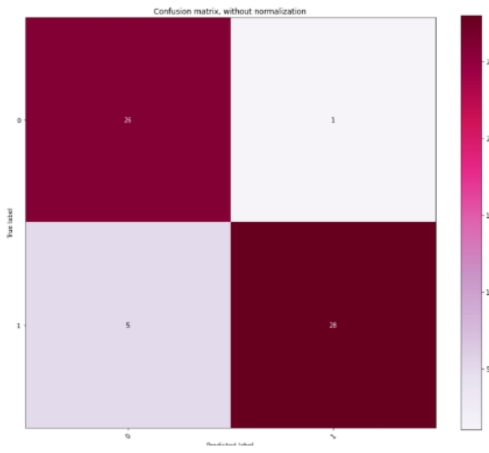


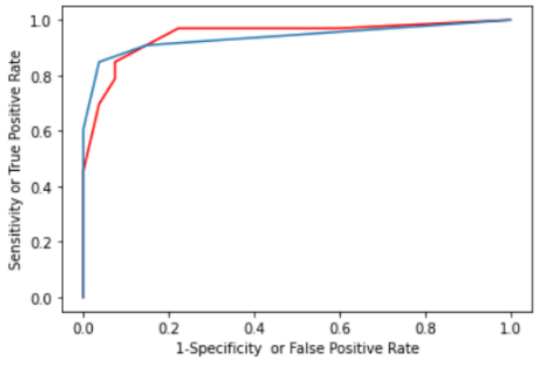
Figure 8: ROC comparison for Naïve Bayes(blue) for a subset of features and KNN (orange) with all features

Figure 9: confusion matrix for Naive Bayes classifier

The number of FN and FP are reduced from 7,2 to 5 and 1, In which is a considerable improvement of the original analysis based on full features.

4.2 For K-Nearest-Neighbors model

We applied the same strategy used in Naïve Bayes subset feature analysis to the KNN model, and we, therefore, obtain the average accuracy of **93.3% (112 features 91.05%)** with best k as **3(112 features 9)** for Repeated k-fold cross-validation, the accuracy of **90% (112 features 86.7%)** for cross-validation on training data split with ratio 7:3, AUC of **0.9377104377104377** (**112 features 0.9438832772166106**),
precison:class=1.0(epithelial)=**0.963(112 features 0.926**),
precision:class=2.0(stromal)=**0.848, (112 features 0.818)**
recall:class=1.0(epithelial)=**0.839(112 features 0.806)**,
recall:class=2.0(stromal)=**0.966 (112 features 0.931)** and
f1-score:class=1.0(epithelial)=**0.897(112 features 0.862)**
f1-score: class=2.0(stromal)=**0.903(112 features 0.871)**,
The ROC (left) and confusion matrix (right):



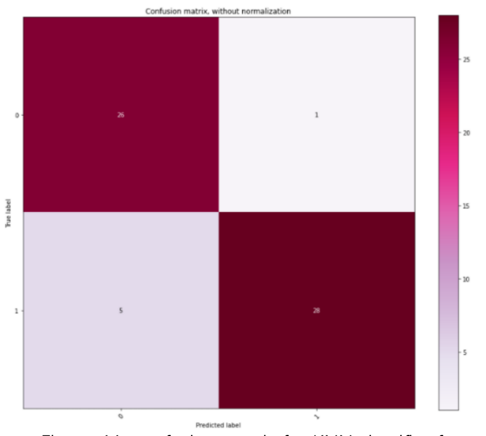


Figure 10: ROC comparison for KNN (blue) Figure 11: confusion matrix for KNN classifier for a

subset of features and KNN (red) with all features

The FN and FP are reduced from 6,2 to 5,1, which is also a decent improvement of the previous entire feature model analysis.

In conclusion, with the evidence provided above, we could deduce that if we choose the subset of features reasonably using feature selection, we will get better performance than the previous classification analysis operated on whole features.

Reference

Brownlee, J., 2020. *How To Choose A Feature Selection Method For Machine Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/> [Accessed 14 November 2020].