

Support Vector Machine:

Support Vector Machine (SVM) is a supervised machine learning algorithm which is used for classification and regression. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin (support vectors), i.e the maximum distance between data points of both classes.

Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. So in short, Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify it with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values $[-1, 1]$ which acts as margin.

Naive Bayes classifier:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The working of the classifier is based on the Bayes theorem.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of A happening, given that B has occurred. Here, B is the evidence and A is the hypothesis. The assumption made here is that the predictors/features are independent. That is, the presence of one particular feature does not affect the other. Hence it is called naive.

Interpretation of the classifiers:

For the classification we have a file named Medreviews.csv. It contains the review of the medicines along with their labels i.e High or low. Then for data pre-processing a normalizer is applied to the reviews which do the following steps :

- 1) It first removes urls, hashtags, special characters.
- 2) It removes all the words having length less than two and all stop words.
- 3) It then lemmatize the text. Lemmatization is the process of converting a word to its base form. It considers the context and converts the word to its meaningful base form.

Since we cannot feed a string or textual data to our machine learning model, so we need to convert it into vectors (numbers) , for that we have used TfidfVectorizer It converts each token (word) to feature index in the matrix, each unique token gets a feature index. Now that we have the right data that we can feed in, to our model. We test the performances of the models using cross validation.

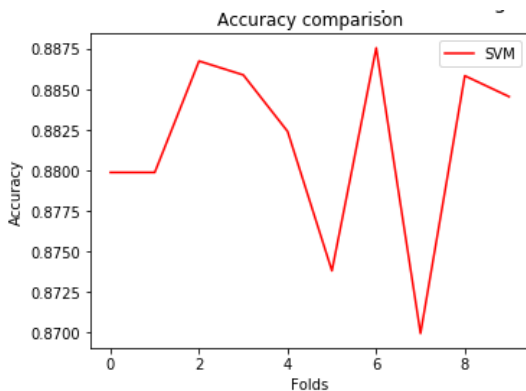
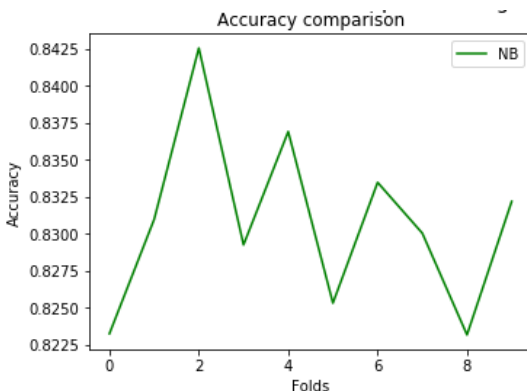
Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. The general procedure is as follows:

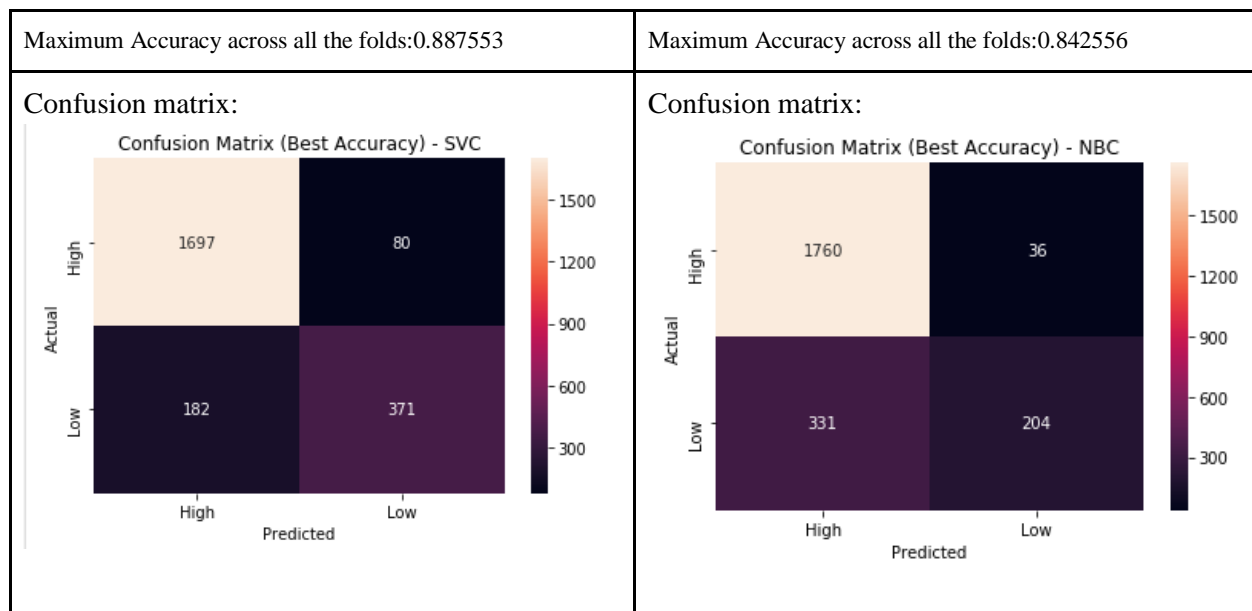
1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
 1. Take the group as a hold out or test data set
 2. Take the remaining groups as a training data set
 3. Fit a model on the training set and evaluate it on the test set
 4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Evaluation of classifiers :

For the evaluation we used two metrics: accuracy and confusion matrix.

1. **Accuracy** is the fraction of predictions our model got right. Formally, **accuracy** has the following **definition: Accuracy** = Number of correct predictions / Total number of predictions.
2. A confusion matrix is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm.
3. Comparison of results:

| Support Vector Machine | Naive Bayes Classifier |
|---|--|
| <p>Accuracy across each fold:</p>  | <p>Accuracy across each fold:</p>  |
| Mean Accuracy of all the folds: 0.881650970 | Mean Accuracy of all the folds:0.8307156 |



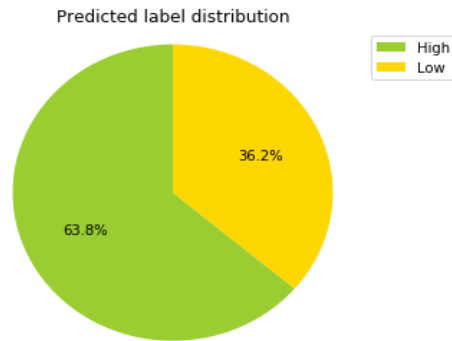
Conclusion:

Both of these algorithms are sensitive to parameter optimization (i.e. different parameter selection can significantly change their output. In the program we didn't optimize the parameters, but we tried K-fold to check accuracy along different folds with their default parameters.

From the experiments, the accuracy on the test data by NB was in range 82%-84% across different folds, and the mean accuracy of the NB was 84. While SVM's accuracy across each fold was in between 87%-88%, with the mean accuracy of 88%. The reason behind this is that Naive Bayes treats all of the features in the data as independent, whereas SVM looks at the interactions between the features to a certain degree, so SVM captures the features in a more better way, hence it is better at classification.

Using SVM for Noratings.csv:

In the task it was given to choose the best classifier to predict ratings in Noratings.csv, So we will be using the Support Vector Machine classifier to classify the reviews given in the No_ratings .csv file. To do this, we fit our model on all of the data that we have in the medreview.csv file, and we fit a tf-idf on the whole data. Then we opened the no rating file and first we cleaned that data with the same normalizer that we used for the cleaning of medreview file. Then the data is transformed into vectors using the saved tf-idf. After that we used the saved SVM model to predict the reviews, and the results for that file is as follows:



PART 2

Understanding a decision tree:

Since all of the classifiers used in this program are formed by assembling of the decision trees. We can think of a decision tree as a series of yes/no questions asked about our data eventually leading to a predicted class. This is an interpretable model because it makes classifications much like we do: we ask a sequence of queries about the available data we have until we arrive at a decision. a decision tree is built by determining the questions (called splits of nodes) that, when answered, lead to the greatest reduction in Gini impurity.

Random Forest :

The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of the observations, splitting nodes in each tree considering a limited number of the features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

AdaBoost:

AdaBoost, short for “Adaptive Boosting”. It focuses on classification problems and aims to convert a set of weak classifiers into a strong one. The weak learners in AdaBoost are decision trees with a single split, called decision stumps. AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.

Gradient Boosting:

Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set.

Gradient Boosting trains many models in a gradual, additive and sequential manner. The major difference between AdaBoost and Gradient Boosting Algorithm is how the two algorithms identify the shortcomings of weak learners (eg. decision trees). While the AdaBoost model identifies the shortcomings by using high weight data points, gradient boosting performs the same by using gradients in the loss function.

Interpretation of the classifiers:

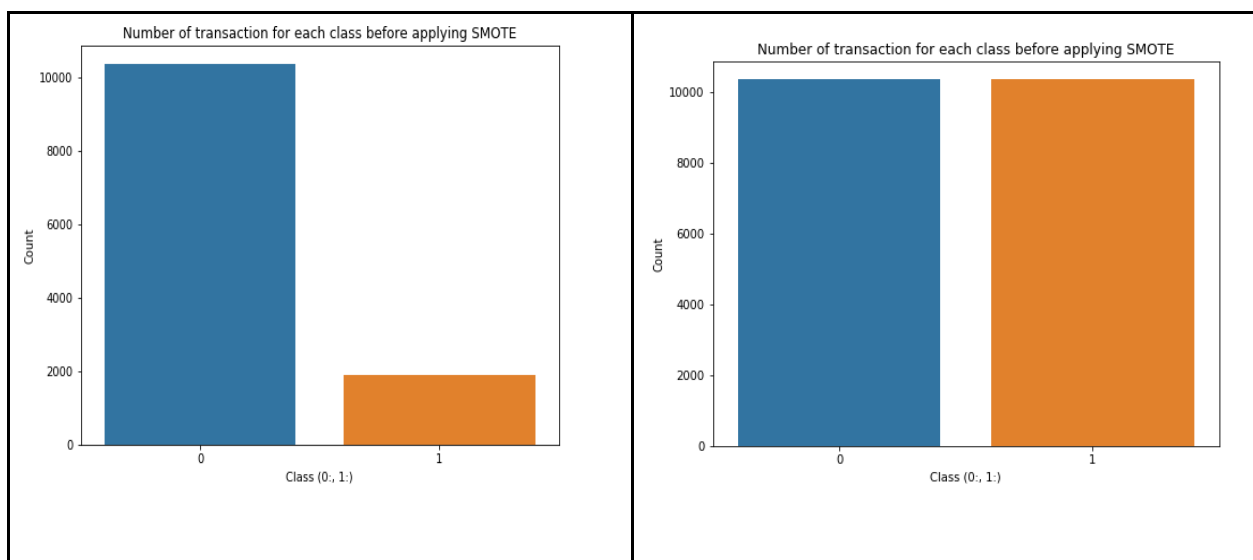
For this problem we have a file E-shop.csv that contains the information that can be used to tell us whether a transaction exists or not. It has 13 feature columns and one target column. We first read the file using pandas in a data frame. The data has some categorical columns, so we converted them into numerical columns, in order to use them.

The problem with our dataset is that our data set is highly imbalanced i.e. a special case for classification problem where the class distribution is not uniform among the classes. Or in simpler terms we have an unequal number of rows for each class label.

We don't train our classifier on this data, because If we have severely imbalanced classes, we can get high overall accuracy without much effort — but without generating any good insights. The overall accuracy might be high, but for the minority class, we will have very low recall.

So to make our model classify better, we use SMOTE technique to generate very similar artificial data.

SMOTE (synthetic minority oversampling technique) is one of the most used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them. **SMOTE** does this by selecting similar records and altering that record one column at a time by a random amount within the difference to the neighboring records.



Evaluating Classifiers

Now that we have a balanced data set, we trained it on the three algorithms. Before training, the data is divided into splits, on one split it is tested, while the rest of the splits are used for training. Then we first used random forest to find out the best feature columns.

How does Random forest choose the best features ?

Random forests consist of 4 –12 hundred decision trees, each of them built over a random extraction of the observations from the dataset and a random extraction of the features. Not every tree sees all the features or all the observations, and this guarantees that the trees are de-correlated and therefore less prone to over-fitting. It takes the result from different trees and checks which of the trees are classifying accurately. Then it chooses the features of those trees as best features.

Hyper tuning the parameters :

After that we hyper tune the parameters of each of the classifier. Hyper parameters tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. We use gridcvsearch algorithm to do this. It combines an estimator with a grid search preamble to tune hyper-parameters. The method picks the optimal parameter from the grid search.

| <u>Random Forest</u> | <u>AdaBoost</u> | <u>Gradient Boosting</u> |
|---|---|---|
| Accuracy:0.911523 | Accuracy:0.89935 | Accuracy:0.91081879 |
| Confusion Matrix: Confusion matrix: [[2790 315] [161 408]] TP: 408 TN: 2790 FP: 315 FN: 161 | Confusion Matrix: Confusion matrix: [[2928 177] [270 299]] TP: 299 TN: 2928 FP: 177 FN: 270 | Confusion Matrix: Confusion matrix: [[2829 276] [189 380]] TP: 380 TN: 2829 FP: 276 FN: 189 |

So from the results above we can clearly state that Random Forest is classifying better than the other algorithms as its accuracy and precision values are greater than the other algorithms.

Recommendations on increasing the number of transactions:

There were almost 13 features in our data set, to predict that a transaction will happen or not. To increase the transactions what we can do is to adjust the values of all those features, so we can get True transactions. So, in the start we did some feature selection i.e. extracting out the main features of our data

set, here is the table showing the importance of each feature.

| | |
|-------------------------------|----------|
| PageValue | 0.364689 |
| Month_Nov | 0.078912 |
| ExitRate | 0.070960 |
| ProductRelated_Duration | 0.064021 |
| ProductRelated | 0.055841 |
| Administrative | 0.048752 |
| Administrative_Duration | 0.043413 |
| BounceRate | 0.040414 |
| Month_May | 0.034383 |
| VisitorType_New_Visitor | 0.033726 |
| VisitorType_Returning_Visitor | 0.031530 |
| Weekend | 0.029463 |
| Month_Mar | 0.020274 |
| Month_Dec | 0.019162 |
| Informational | 0.018681 |
| Informational_Duration | 0.015804 |
| Month_Oct | 0.007736 |
| Month_Sep | 0.007703 |
| Month_Jul | 0.004753 |
| Month_Aug | 0.004214 |
| SpecialDay | 0.003387 |
| Month_June | 0.001834 |
| Month_Feb | 0.000346 |

From the above table we can see that the transaction column is mainly depending upon the PageValue column, also in our data we observed that whereas the page value is greater, the transaction is True. So, to get a lot of transactions we need to increase our page value, the higher the page value, more will be the transactions.

As far as choosing the best algorithm is concerned, we trained all of the algorithms on the first 5 selected features. And noted out their accuracies, the random forest was performing better than the rest of the algorithms, also we know that random forest is prone to overfitting, so we choose it as our classifier to predict when we can get a transaction.