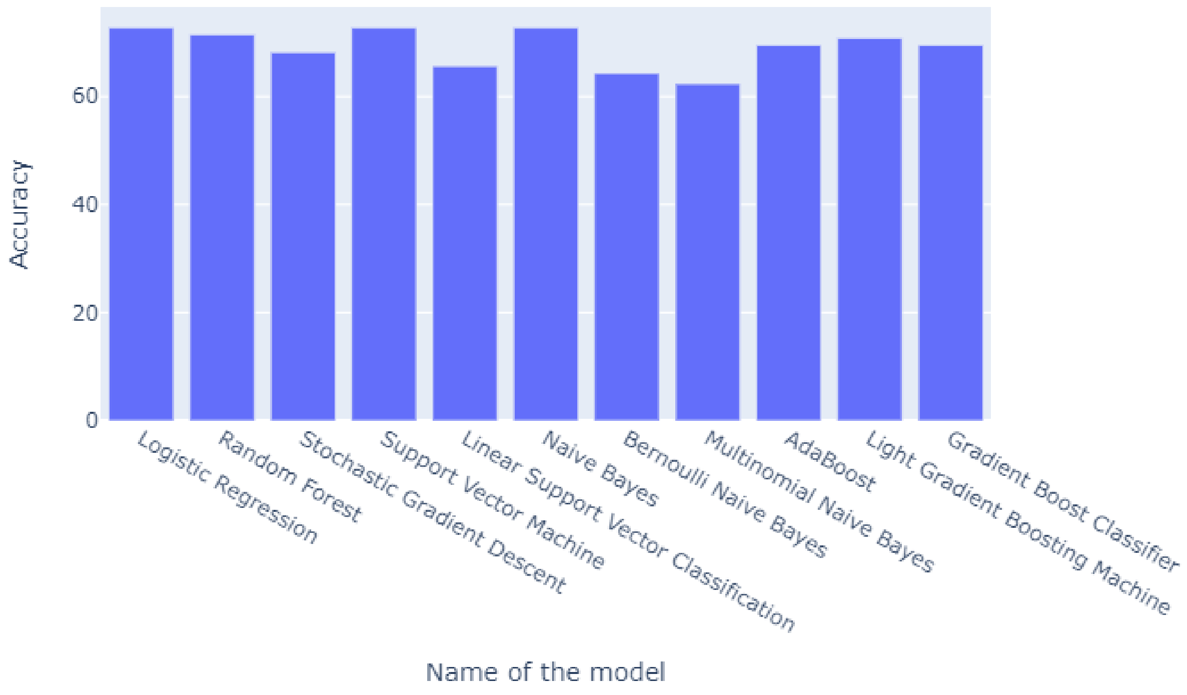
Diabetes is a type of chronic disease which is more common among people of all age groups. Predicting this disease at an early stage can help a person to take the necessary precautions and change his/her lifestyle accordingly to either prevent the occurrence of this disease or control the disease(For people who already have the disease).

1. Prepare the data set using several methods to train the model.
2. Build a model which can give high accuracy in predicting the disease.

Dataset:
This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Implementation:
The following 11 Machine Learning algorithms were trained on the dataset mentioned above:

- Logistic Regression
- Random Forest Classifier
- Stochastic Gradient Descent Classifier
- Support Vector Machine
- Linear Support Vector Machine
- Gaussian Naive Nayes
- Binomial Naive Bayes
- Multinomial Naive Bayes
- AdaBoost Classifier
- Light Gradient Boosting Machine
- Gradient Boosting Machine

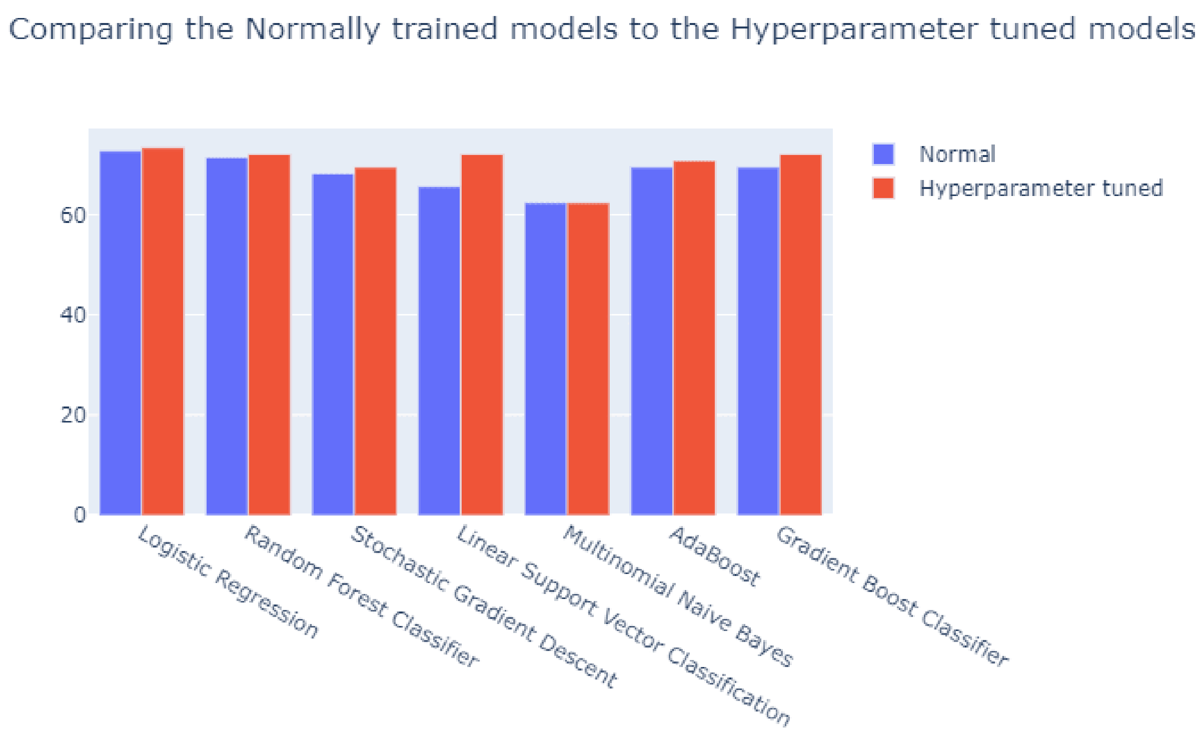


Model	Accuracy
Logistic Regression	72.727
Random Forest Classifier	71.429
Stochastic Gradient Descent Classifier	68.182
Support Vector Machine	72.727
Linear Support Vector Machine	65.584
Gaussian Naive Bayes	72.727
Bernoulli Naive Bayes	64.286
Multinomial Naive Bayes	62.338
AdaBoost Classifier	69.481
Light Gradient Boosting Machine	70.779
Gradient Boosting Classifier	69.481

Closely looking at the models and their accuracy levels, I decided to tune the hyperparameters using GridSearchCV hoping to get higher levels of accuracy. GridSearchCV takes a dictionary of parameters as an input and then forms various combinations of the various parameters and trains the model using each of those parameter sets and outputs the parameter set which gives the highest accuracy.
Not all models could be hyperparameter tuned either because of computational constraints or because they didn't have enough parameter options, so the following 7 models were chosen:

- Logistic Regression
- Random Forest Classifier
- Stochastic Gradient Descent Classifier
- Linear Support Vector Machine
- Multinomial Naive Bayes
- AdaBoost Classifier
- Gradient Boosting Machine

The accuracy of the hyperparameter tuned models wasn't a lot more than the models previously trained.



Model	Accuracy	Change in Accuracy	Parameter dictionary	Hyper parameters	Default parameters
Logistic Regression	73.377	0.65	{ 'C': [0.1, 1, 10], 'max_iter': [100, 150, 200, 250, 500] 'n_jobs': [-1], 'penalty': ['l1', 'l2', 'elasticnet'], 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'] }	{ 'C': 1, 'max_iter': 100, 'n_jobs': -1, 'penalty': 'l2', 'solver': 'newton-cg' }	{ ‘C’: 1.0, ‘max_iter’: 100, ‘n_jobs’: None, ‘penalty’: ‘l2’, ‘solver’: ‘lbfgs’ }
Random Forest Classifier	72.078	0.649	{ 'criterion': ['gini', 'entropy'], 'max_features': [3,5,7, 'auto', 'sqrt', 'log2'], 'n_estimators': [100, 150, 200, 250, 500] }	{ 'criterion': 'gini', 'max_features': 'log2', 'n_estimators': 100 }	{ ‘criterion’: ‘gini’, ‘max_features’: ‘auto’, ‘n_estimators’: 100 }
Stochastic Gradient Descent Classifier	69.481	1.299	{ 'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000], 'loss': ['hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'], 'n_jobs': [-1] 'penalty': ['l1', 'l2', 'elasticnet'] }	{ 'alpha': 100, 'loss': 'squared_hinge', 'n_jobs': -1, 'penalty': 'l2' }	{ ‘alpha’: 0.0001, ‘loss’: ‘hinge’, ‘n_jobs’: None, ‘penalty’: ‘l2’ }
Linear Support Vector Machine	72.078	6.494	{ 'C': [1, 10, 100], 'loss': ['hinge', 'squared_hinge'] 'multi_class': ['ovr', 'crammer_singer'] 'penalty': ['l1', 'l2'], }	{ 'C': 1, 'loss': 'squared_hinge', 'multi_class': 'crammer_singer', 'penalty': 'l1' }	{ ‘C’: 1.0, ‘loss’: ‘squared_hinge’, ‘multi_class’: ‘ovr’, ‘penalty’: ‘l2’ }
Multinomial Naive Bayes	62.338	0	{ 'alpha': [1, 10], 'fit_prior': ['True', 'False'] }	{ 'alpha': 1, 'fit_prior': 'True' }	{ ‘alpha’: 1.0, ‘fit_prior’: ‘True’ }
AdaBoost	70.779	1.298	{ 'algorithm': ['SAMME', 'SAMME.R'],	{ 'algorithm': 'SAMME',	{ ‘algorithm’: ‘SAMME.R’,

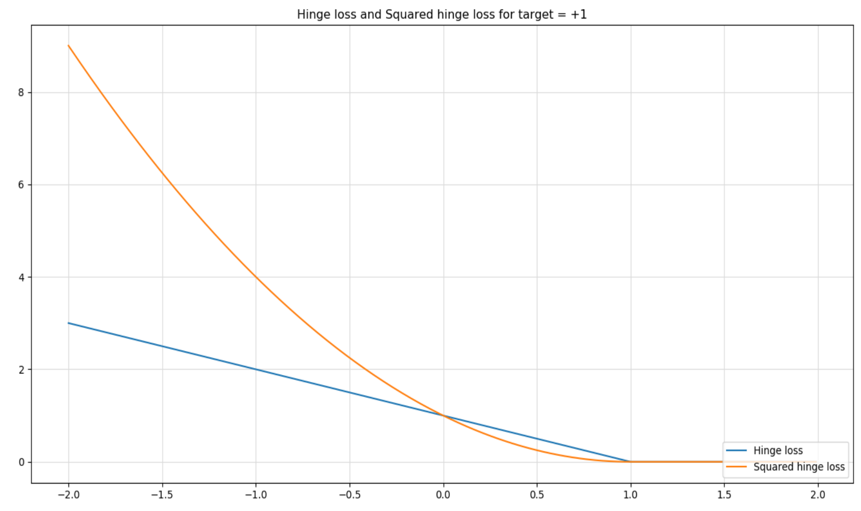
Classifier			'n_estimators': [1, 10, 50, 100, 200, 500, 1000] }	'n_estimators': 200 }	‘n_estimators’: 50 }
Gradient Boosting Machine	72.078	2.597	{ 'criterion': ['friedman_mse', 'squared_mse', 'mse', 'mae'], 'loss': ['deviance', 'exponential'], 'learning_rate':[0.1, 1, 10, 100], 'n_estimators':[1, 10, 100, 1000] }	{ 'criterion': 'friedman_mse', 'learning_rate': 1, 'loss': 'exponential', 'n_estimators': 10 }	{ ‘criterion’: ‘friedman_mse’, ‘learning_rate’: 1.0, ‘loss’: ‘deviance’, ‘n_estimators’: 100 }

The minuscule increases in the accuracy of the Logistic Regression and Random Forest Classifier can be credited to the fact that the majority of the parameters selected after the tuning were similar to the default parameters.

The only change in the Logistic Regression model was the ‘solver’ parameter which chose ‘newton-cg’ as its value instead of the default ‘lbfgs’. Both the methods do quadratic approximations at every point x. For this, it needs a gradient and a Hessian. If the Hessian is not given, it must be calculated by the algorithm. ‘Newton-cg’ calculates this inverse Hessian explicitly whereas ‘lbfgs’ calculate an approximation to the Hessian (i.e., the curvature) based on the gradient, which is a faster operation. This exact calculation of the Hessian might be one of the reasons why ‘newton-cg’ gave better accuracy and was thus chosen. ‘Lbfgs’ is also the lower memory version of ‘bfgs’ that stores far less memory at every step what this means is that the algorithms cannot find the true minimum.

Random Forest Classifier chose to with ‘log2’ instead of ‘auto’ for its ‘max_features’ parameter which can to attributed to the fact that ‘max_features’ is the size of the random subsets of features to consider when splitting a node i.e. it is the number of features that are considered on a per-split level, rather than on the entire decision tree construction. When ‘max_features’= ‘auto’ no feature subset selection is performed in the trees, so the Random Forest is actually a bagged ensemble of ordinary regression trees. ‘Log2’ gives us a superior result and so it has been selected as the parameter value for ‘max_features’.

The increase in the accuracy of Stochastic Gradient Descent was almost double of the two algorithms we have discussed up till now. The ‘alpha’ value after hyperparameter tuning was a million times that of the default parameter. ‘Alpha’ is the value that multiplies the regularization term. The higher the value, the stronger the regularization. Setting ‘n_jobs’ to -1 means the algorithm uses all the CPU available. The default value of the ‘loss’ parameter is ‘hinge’ but after hyperparameter tuning, it selected ‘squared_hinge’. ‘Squared_hinge’ punishes larger errors significantly more than the smaller errors. The following graph shows the comparison between the loss functions of ‘hinge’ and ‘squared_hinge’



Linear Support Vector Machine has the highest increase in accuracy after hyperparameter tuning. ‘Multi_class’ and ‘penalty’ are the two parameters that have different values as compared to the default. Selection of ‘l1’ for the ‘penalty’ leads to ‘coef_’ vectors (weights assigned to the features) that are sparse. ‘Crammer_singer’ optimizes a joint objective over all classes.

The AdaBoost Classifier found that the model had better predictions when the ‘n_estimators’ value was high.

Gradient Boosting Machine algorithm had results quite the opposite of AdaBoost as it used a ‘n_estimators’ value lower than the default. Furthermore, it chose the ‘exponential’ loss function as compared to the conventional ‘deviance’ because the difference between them is in degree. The penalty associated with binomial deviance increases linearly, whereas the exponential criterion increases the influence of such observations exponentially. Not to say that exponential loss will be more sensitive to outliers.

Multinomial Naive Bayes had no increase in accuracy after hyperparameter tuning as the parameter values are chosen are the same as the default parameter values.

A neural network was also trained on the data. The summary of the network is given below

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 8)                 0
_________________________________________________________________
dense (Dense)                (None, 128)               1152
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 130
=================================================================
Total params: 9,538
Trainable params: 9,538
Non-trainable params: 0
_________________________________________________________________
```

The front end was deployed using Streamlit. The user has to input eight values in their respective fields and press submit, those values are then fed to the ten Machine Learning algorithms, and the neural network. The outputs from all these are stored in a list and compared. The value that occurs the most number of times will be displayed to the user.

Frontend Link: https://share.streamlit.io/jashtailor/exposys/main/mk2.py