# Regression Models

Brief comparison of the main regression algorithms in scikit-learn:

1. Random Forest Regression: An ensemble algorithm that uses multiple decision trees to make a prediction. Random forest combines the predictions of multiple trees to produce a more robust and accurate result, and it is less prone to overfitting compared to decision trees.
2. Support Vector Regression (SVR): A non-linear algorithm that uses a technique called kernel trick to transform the input data into a higher-dimensional space. SVR is suitable for problems with complex relationships between the independent and dependent variables, but it can be slow to train and may not scale well to large datasets.
3. Gradient Boosting Regression: An ensemble algorithm that builds a series of weak learners and combines their predictions to produce a final result. Gradient boosting can handle both linear and non-linear relationships, and it is often considered one of the best regression algorithms in terms of accuracy. However, it can be slow to train and may overfit the data if the number of trees is too large.
4. Adaboost Regression: An ensemble algorithm that builds a series of weak learners and gives more weight to instances that are misclassified by the previous weak learners. Adaboost can handle both linear and non-linear relationships, and it is fast to train. However, it can be prone to overfitting if the number of weak learners is too large.
5. ElasticNet: A linear regression algorithm that combines the penalties of L1 and L2 regularization, which help to prevent overfitting and ensure stability. ElasticNet can handle highly correlated features and sparse data, but it may not perform well on non-linear problems.
6. SGDRegressor: A linear regression algorithm that uses the Stochastic Gradient Descent optimization method to find the optimal coefficients. SGDRegressor is suitable for large-scale problems and online learning, but it can be sensitive to the choice of hyperparameters.
7. XGBRegressor: An implementation of gradient boosting regression using the XGBoost library. XGBRegressor is optimized for speed and scalability, and it provides a number of advanced features such as parallel computing and automatic tuning of hyperparameters.
8. LGBMRegressor: An implementation of gradient boosting regression using the LightGBM library. LGBMRegressor is designed for fast training and is suitable for large-scale problems. It provides a number of advanced features such as parallel computing and handling of missing values.
9. BayesianRidge: A linear regression algorithm that uses Bayesian techniques to estimate the coefficients. BayesianRidge provides regularization to prevent overfitting and is suitable for problems with sparse data and many features.
10. CatBoostRegressor: An implementation of gradient boosting regression using the CatBoost library. CatBoostRegressor is designed to handle categorical features natively and provides advanced features such as automatic handling of missing values and parallel computing.
11. Kernel Ridge Regression (KernelRidge): A non-linear regression algorithm that uses a kernel trick to transform the input data into a higher-dimensional space. KernelRidge is suitable for problems with complex relationships between the independent and dependent variables, but it may be slow to train and may not scale well to large datasets.

It's important to note that the performance of these algorithms depends heavily on the specific problem and data, so it's advisable to try several algorithms and compare their performance using appropriate evaluation metrics.

Comparison table:

|  |  |  |
| --- | --- | --- |
| Algorithm | Pros | Cons |
| Random Forest Regression | * Can handle non-linear relationships and high-dimensional feature spaces. * Is relatively fast to train and easy to use. * Is robust to overfitting and handles missing values well. | * Can be slow to make predictions for large datasets. * Can be difficult to interpret the results due to the complexity of the algorithm. |
| SVR | * Can handle complex relationships between the independent and dependent variables. * Is highly flexible and can model non-linear relationships. | * Can be slow to train and may not scale well to large datasets. * Can be sensitive to the choice of kernel function and regularization parameters. |
| ElasticNet | * Can handle highly correlated features and sparse data. * Combines the penalties of L1 and L2 regularization, which help to prevent overfitting and ensure stability. | * May not perform well on non-linear problems. * Can be sensitive to the choice of regularization parameters. |
| SGDRegressor | * Suitable for large-scale problems and online learning. * Is relatively fast to train and easy to use. | * Can be sensitive to the choice of hyperparameters. * May not perform well on non-linear problems. |
| XGBRegressor | * Optimized for speed and scalability. * Provides a number of advanced features such as parallel computing and automatic tuning of hyperparameters. | * Can be sensitive to the choice of hyperparameters. * Can overfit the data if the number of trees is too large. |
| LGBMRegressor | * Designed for fast training and is suitable for large-scale problems. * Provides a number of advanced features such as parallel computing and handling of missing values. | * Can be sensitive to the choice of hyperparameters. * Can overfit the data if the number of trees is too large. |

## RandomForestRegressor

A Decision Tree is a popular type of supervised learning algorithm that is used for both classification and regression problems. It works by building a tree-based model that predicts the value of a target variable based on the input features of the data. The model is represented as a tree structure, with each node representing a feature and each branch representing a decision rule. The root node, which is located at the top of the tree, represents the initial feature that splits the data into smaller subsets. The subsequent branches are created based on the next best feature that best separates the data into its target classes or values.

Each internal node in the tree splits the data based on a feature, and each branch represents a possible outcome of that feature. The decision rules are simple tests that determine whether the value of a feature is above or below a certain threshold. When a branch leads to a leaf node, the prediction is made based on the target class or value assigned to that node.

Random Forest Regressor is an extension of the Decision Tree algorithm, but instead of relying on a single tree, it uses an ensemble of multiple trees to make predictions. This is known as an ensemble learning approach. In this method, a random subset of features is selected for each tree in the ensemble, and a separate Decision Tree is built for each subset. The predictions of each individual tree are then averaged to produce the final prediction. This method helps to reduce overfitting, which is a common problem with the Decision Tree algorithm, and it also improves the accuracy of the model by reducing the variance in the predictions.

## XGBRegressor

Gradient Boost Regressor is a type of ensemble learning algorithm that leverages multiple "weak learners" (such as decision trees) to build a robust predictive model. The algorithm operates by adding weak learners one by one in a sequential manner, where each new learner corrects the mistakes made by its predecessors. Initially, a basic model is fitted to the data and then, subsequent models are gradually added to reduce the residual errors from previous models. The final model is a composite of all weak learners, with each learner contributing proportionally to its accuracy.

Gradient Boost Regressor is a well-known gradient boosting algorithm that fits weak learners in a sequential manner to improve the overall accuracy of the model. It uses the gradient descent optimization method to minimize the loss function and improve the performance of the model.

On the other hand, XGradBoost Regressor is an extension of Gradient Boost Regressor that aims to improve its scalability and efficiency. It uses a distributed computing architecture and parallel processing to perform gradient boosting on large-scale datasets. It can also handle missing values and noisy data, making it suitable for use in real-world scenarios. XGradBoost Regressor can also perform feature selection, reducing the number of features and increasing the interpretability of the model.

## SVR

Support Vector Regression (SVR) is a supervised learning technique that uses Support Vector Machines (SVMs) to forecast continuous values. It accomplishes this by transforming the input data into a high-dimensional feature space, where the data points can be separated even if they are not linearly separable in the original feature space. To determine this hyperplane, the algorithm finds the optimal boundary that maximizes the margin between the two classes in the feature space. This hyperplane is then used to predict the target values for new, unseen data points.

## Elastic Net

ElasticNet is a method used in machine learning to regularize regression models, meaning it helps prevent overfitting and improves the model's ability to generalize to new data. It combines the strengths of two popular regularization methods, Lasso and Ridge regression.

Lasso regularization adds a penalty to the absolute values of the coefficients, while Ridge regularization adds a penalty to the squares of the coefficients. ElasticNet combines these two penalties by adding a penalty to the sum of both the absolute values and the squares of the coefficients. This combination allows ElasticNet to select important features while shrinking the coefficients towards zero.

The strength of the regularization is determined by a single parameter, which balances the influence of the L1 and L2 penalties. By adjusting this parameter, it is possible to control the trade-off between feature selection and coefficient shrinkage.

ElasticNet can be applied to both linear and non-linear regression problems, making it a versatile and powerful regularization technique.

## SGD Regressor

Stochastic Gradient Descent (SGD) Regressor is a machine learning algorithm used to perform regression, a type of prediction where the goal is to estimate a continuous value. It is an iterative algorithm, meaning it repeats a process multiple times until a stopping criterion is met.

SGD Regressor works by optimizing a cost function using gradient descent. The cost function measures how well the model fits the training data and is used to guide the optimization process. During each iteration, a random subset of the training data is selected and the gradient of the cost function with respect to the model's parameters is computed. This gradient tells us the direction in which we should update the parameters in order to reduce the cost.

The parameters are updated using the computed gradient and the process is repeated until the cost function converges to a minimum, meaning that it can no longer be significantly reduced by further updates to the parameters. At this point, the optimization process is considered to be complete and the model is ready for use.

SGD Regressor can be applied to both linear and non-linear regression problems and is particularly useful when working with large datasets where it can be computationally infeasible to compute the gradient using all the data.

## LGBM Regressor

LightGBM Regressor is a machine learning algorithm used to perform regression, a type of prediction where the goal is to estimate a continuous value. It is an open-source gradient boosting framework, meaning that it is a freely available tool for building machine learning models. LightGBM Regressor uses a tree-based learning algorithm, meaning it builds a model by constructing a series of decision trees.

The algorithm works by first fitting a simple tree-based model to the data, and then iteratively adding new trees that correct the residual errors of the previous trees. This process is repeated until a stopping criterion is met, such as a maximum number of trees or a minimum improvement in the cost function.

The final model is a weighted combination of all the trees, where each tree contributes to the prediction based on its accuracy. LightGBM Regressor is efficient and scalable, making it well-suited for large datasets and high-dimensional data. It can be used for both linear and non-linear regression problems.

# OneHotEncoding

One-hot encoding is a technique used to represent categorical data in a numerical format so that it can be used in machine learning algorithms. The process of one-hot encoding involves converting each unique category in a categorical feature into a new binary feature (i.e., a feature that takes on the value 0 or 1) in a new data set.

For example, suppose you have a categorical feature Color with three unique categories: Red, Green, and Blue. In a one-hot encoding of this feature, you would create three new binary features, one for each unique category: Red\_encoded, Green\_encoded, and Blue\_encoded. The value in each new feature would be set to 1 if the original Color feature had the corresponding category, and 0 otherwise.

The resulting data set would have the same number of instances (i.e., rows) as the original data set, but would have one additional feature for each unique category in the original categorical feature. This encoding allows machine learning algorithms to work with categorical data, as the data is now represented in a numerical format.

# Evaluation Metrics

\*

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Description** | **Advantage** | **Disadvantage** |
| Mean Absolute Error (MAE) | Measures the average magnitude of errors in a set of predictions, without considering their direction. | It is relatively simple to understand and interpret. | It does not provide any information about the direction of the error. |
| Root Mean Squared Error (RMSE) | Measures the average magnitude of errors in a set of predictions, taking into account their direction. | It provides information about the direction of the error. | It is more difficult to interpret than MAE. |
| R-squared | Measures the proportion of variance in the dependent variable that is explained by the independent variables. | It provides a measure of the goodness of fit of the model. | It can be misleading when the number of independent variables is large, as it tends to increase even when the model does not fit the data well. |
| Adjusted R-squared | Similar to R-squared, but takes into account the number of independent variables in the model. | It provides a more accurate measure of the goodness of fit of the model, especially when the number of independent variables is large. | It can be more difficult to interpret than R-squared. |

* R2: In general, an R2 value of 1 indicates a perfect fit between the observed data and the model's predictions, while an R2 value of 0 indicates that the model doesn't explain any of the variation in the data. Values between 0 and 1 indicate the proportion of the variation in the data that is explained by the model. In practice, a value of R2 greater than 0.7 is considered a good fit.
* MAE: The MAE value gives an idea of the average error in the model's predictions. A lower MAE indicates that the model's predictions are closer to the actual values. There is no set "good" value for MAE as it depends on the problem and the units of the response variable.
* RMSE: Similar to MAE, the RMSE provides a measure of the average error of the model's predictions. However, RMSE gives more weight to larger errors, so it is more sensitive to outliers than MAE. A lower RMSE value indicates that the model's predictions are closer to the actual values.
* Adjusted R2: The adjusted R2 value is a modified version of R2 that takes into account the number of predictors in the model and the sample size. The adjusted R2 value gives a better indication of the model's performance than R2, especially in models with many predictors. A value of adjusted R2 greater than 0.7 is considered a good fit.

# Bibliography

Here is a list of relevant literature for the various machine learning algorithms you mentioned:

1. Gradient Boost Regressor:
   * J.H. Friedman. "Stochastic Gradient Boosting." (1999)
   * T. Hastie, R. Tibshirani, and J. Friedman. "The Elements of Statistical Learning." (2009)
   * Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. Annals of statistics, 1189-1232.
   * Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).
2. Support Vector Regression (SVR):
   * V. Vapnik. "The Nature of Statistical Learning Theory." (1995)
   * C. Chang and C. Lin. "LIBSVM: A Library for Support Vector Machines." (2011)
   * Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J., & Vapnik, V. (1997). Support vector regression machines. Advances in neural information processing systems, 155-161.
   * Suykens, J. A., Vandewalle, J., & De Moor, B. (2002). Least squares support vector machine classifiers. Neural processing letters, 953-961.
3. Adaboost Regressor:
   * Y. Freund and R. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." (1997)
   * J. Zhu, H. Zou, S. Rosset, and T. Hastie. "Multi-Class AdaBoost." (2009)
4. Random Forest Regressor:
   * Breiman, L. (2001). Random forests. Machine learning, 45(1), 5-32.
   * Cutler, A., & Breiman, L. (2007). Random forests for classification in ecology. Ecology, 88(11), 2783-2792.
   * A. Criminisi, J. Shotton, and E. Konukoglu. "Regression forests for efficient anatomy detection in crowded scenes." (2012)
5. ElasticNet:
   * Z. H. Zhou and X. X. Liu. "Regularization and Variable Selection via the Elastic Net." (2004)
   * Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.
   * Fan, J., & Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American Statistical Association, 96(456), 1348-1360.
6. Stochastic Gradient Descent Regressor (SGDRegressor):
   * Y. Bottou. "Stochastic gradient descent tricks." (2012)
   * L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization Methods for Large-Scale Machine Learning." (2018)
   * Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. Proceedings of COMPSTAT'2010, 177-186.
   * Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.
7. LightGBM Regressor:
   * Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Advances in neural information processing systems (pp. 3149-3157).
   * Chen, T., & He, X. (2019). XGBoost and LightGBM: A comparative study for regression tasks. arXiv preprint arXiv:1907.08789.
   * T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System." (2016)
8. Scikit-learn Library
   * Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.
9. Evaluation Metrics:
   * An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani
   * Pattern Recognition and Machine Learning by Christopher Bishop
   * Machine Learning Mastery by Jason Brownlee
   * Hands-On Machine Learning with Scikit-Learn and TensorFlow by Aurélien Géron
   * Applied Predictive Modeling by Max Kuhn and Kjell Johnson

# App Development

Here's an outline of the steps you would need to follow:

1. Train your scikit-learn model in Python and save it to a pickled file, using the pickle module:

python

import pickle

# Train your model

model = ...

# Save the model to a file

with open('model.pkl', 'wb') as f:

pickle.dump(model, f)

1. In your React Native app, use the react-native-fs package to read the pickled file from the device's local storage:

javascript

import RNFS from 'react-native-fs';

const modelPath = RNFS.DocumentDirectoryPath + '/model.pkl';

async function loadModel() {

const modelFile = await RNFS.readFile(modelPath, 'binary');

const model = pickle.loads(