

# Interactive DATA1030 Visualizations: Practicum Report

## Project Overview

This practicum project involved the creation of interactive visualizations for machine learning classification and gradient descent. These visualizations will supplement the DATA1030 curriculum and aim to help students understand how different ML classification algorithms work, how hyperparameters shape predictions, bias-variance trade-offs, and gradient descent. This project was supervised by Professor Andras Zsom.

## Project Deliverables

### Deliverable 1: Interactive visualizations of ML classification models

The main project deliverables for this practicum were interactive DATA1030 visualizations of classification ML models with variable hyperparameters. In particular, I created functions to plot interactive visualizations that are generalizable to any classification ML model with any hyperparameter grid. These visualizations have sliders for each hyperparameter which update a contour plot of predicted probabilities as well as the model's decision boundary for given input data. The goal of these deliverables are to help DATA1030 students understand how different ML classification models work and how the classifications vary across different hyperparameters.

In particular, the main project deliverables include interactive plots for the following classification ML models (with any given hyperparameters):

- Logistic Regression
- Random Forest
- Support Vector Machine (SVM)
- XGBoost
- K-Nearest Neighbors (KNN)

### Deliverable 2: Interactive visualizations of gradient descent

The secondary deliverable includes a 2-plot interactive visualization of gradient descent as a function of learning rates. In particular, the first plot illustrates the gradient descent path over a

contour plot of the cost function, and the second plot shows the cost history over the gradient descent iterations. For these plots, a single slider controls the learning rate.

## Milestones

Before the first milestone, the first tasks were to set up the Conda environment and to research the best tools to plot interactive plots in Python. I examined Plotly, Plotly Express, and Ipywidget and selected Ipywidgets and Plotly's Graph Objects to create interactive plots with multiple sliders (for each hyperparameter).

### Milestone 1: Create function to generate data for plotting

The first milestone was to create a function that generates data for the interactive plot from a given ML model, input data, and range of hyperparameters. It outputs a dictionary containing 2D arrays of predicted probabilities (based on the model's classification).

Inputs:

- input data X
- input data y
- ML algorithm
- 2 hyperparameter ranges
- contour plot grid step size 'h'

Output:

- dictionary of 2D arrays of predicted probabilities

### Milestone 2: Create function to interactively plot ML classification for 2 hyperparameters

The second milestone was to create a function to interactively plot the classification data generated by the first function. As a starting point, this function plots the predicted probability contour plot (with a decision boundary for 0.5 probability) against a scatter plot of input data for 2 given hyperparameters. Thus, this plot has two sliders to adjust the hyperparameter values, which update the contour plot and decision boundary.

This function takes in the generated dictionary of 2D predicted probability arrays and the input data (for the scatter plot). Thus, it has the following inputs and outputs:

Inputs:

- input data X
- input data y
- 2 hyperparameter ranges
- dictionary of 2D arrays of predicted probabilities
- contour plot grid step size 'h'

Output:

- interactive figure with a slider for both hyperparameters controlling the contour plot of predicted classification probabilities on top of a scatterplot of the input data

### Milestone 3: Generalize functions to $n$ hyperparameters

The third milestone was to generalize the data generation and plotting functions to  $n$  hyperparameters. Thus, the data generation function takes in a hyperparameter grid and outputs a dictionary of 2D arrays that are accessible by hyperparameter combination. In addition, the plotting function takes in the hyperparameter grid and creates a variable number of sliders to update the contour plot and decision boundary.

It has the following inputs and outputs:

Inputs:

- input data X
- input data y
- hyperparameter grid
- dictionary of 2D arrays of predicted probabilities
- contour plot grid step size 'h'

Output:

- interactive figure with sliders for a variable number of hyperparameters controlling the contour plot of predicted classification probabilities on top of a scatterplot of the input data

### Milestone 4: Create gradient descent interactive plot

The fourth milestone was to create interactive plots for gradient descent as a function of learning rates. The first plot illustrates the gradient descent path over a contour plot of the cost function, and the second plot shows the cost history over the gradient descent iterations.

The function to generate the gradient descent path data takes in the input data, the vector of random initial thetas, and vector of learning rates, and outputs a dictionary of theta history, cost history, and final thetas stored by learning rate.

The plotting function takes in the generated theta and cost histories, cost function, and vector of learning rates and outputs a side-by-side interactive plot of the gradient descent path over the contour plot of the cost function, as well as a line plot of the cost history over the number of iterations. The slider updates the learning rate and the resulting gradient descent path and cost history in each plot.

## DSI Competencies

### 1. Apply Probability, Statistics, and Machine Learning to Real-World Problem

This project visualizes classification probabilities for a range of machine learning models. In particular, this project examines how predicted probabilities and decision boundaries shift as model hyperparameters change. As such, this project directly applies probability, statistics, and machine learning to classification problems.

While the classification data is toy data, the project illustrates the strengths and limitations of machine learning models to prepare students to apply them appropriately in real-world situations. Being able to visualize how different models and parameters shape classification interactively provides an intuitive understanding of how they work as well as of critical concepts like the bias-variance trade-off.

Understanding how ML models classify and their pros and cons can help determine which ML model to use for a given classification problem. Thus, this project's visualizations might help students decide whether a given model is appropriate and select hyperparameters for a good bias-variance trade-off. For example, the random forest classification plot shows a step-like decision boundary that can capture nonlinear dependencies but yields step-like predictions that are extrapolated outwards for outliers. Thus, students who prefer less certainty for outliers in their data and smoother predictions might opt for SVM instead.

The classification visualizations directly show how the predicted probabilities of various classification algorithms shift as a function of model hyperparameters. As such, they provide

intuitive insight into how hyperparameters (such as SVM's regularization and gamma kernel coefficient, KNN's number of nearest neighbors, and XGBoost's regularization and maximum depth) shape the model's predicted probabilities within the given grid. For example, the changing the KNN's weighting method from uniform to distance-based weights clearly illustrates how the decision boundary more closely wraps around data points while giving less weight to outliers. Moreover, lowering the number of neighbors clearly increased the model certainty - visually shown by the saturation of the contour plot (dark blue or red) - as fewer and less mixed neighbors 'vote' on class membership.

The gradient descent plots directly show how the gradient descent optimization algorithm works, illustrating the gradient descent path over the given cost function and how the cost decreases over the iterations. Moreover, the plots show how these change as a function of learning rate (which shapes how big the steps in the opposite direction of the gradient are). In particular, they show how quickly the path and cost converge to the minimum and for what learning rates they begin to diverge. For example, for the given data, the gradient descent path begins to diverge for a learning rate of 1.9, whereby the steps overshoot and drive the cost up rapidly.

Overall, the classification visualizations illustrate different ML algorithms' classifications as a function of hyperparameters, their behavior with respect to outliers, nonlinearity, prediction smoothness, and interpretability. In addition, the gradient descent plots show how the gradient descent path and cost history vary as a function of learning rate, and when the algorithm fails to converge.

## 2. Formulate Questions, Analyze, and Draw Conclusions from Big Data Sets

Since this project aims to create interactive classification plots as educational materials, it uses a toy 2d binary classification dataset chosen for its ease of visualization and nonlinear dependencies. Particularly, the dataset has 2 target classes with 2 features in order to plot the data clearly and in 2D. As such, this project does not use big datasets.

However, this project aims to address the questions, for each major classification ML model, of how changes in a given hyperparameter impact the resulting classification (probabilities and decision boundary). By accepting any ML classification model and range of hyperparameters, this project creates a tool to directly analyze such questions. From this, conclusions about the behavior and applicability of different ML models can be drawn.

In addition, the gradient descent plots aim to address several questions - how does the gradient descent path vary as a function of learning rate, how does the cost decrease as a function of learning rate, and for what values of learning rate does the gradient descent fail to converge? Thus, the gradient descent plots create a tool to answer these questions visually.

### 3. Connect Limitations in Data and Analysis to Bias in Results

In this project, there are both limitations in the data used and in the ML models used for analysis. The interactive plots illustrate each ML model's limitations, which are important to understand when applying these models to real-world data. These limitations include applicability for large datasets (for example, SVM is not suitable as it has a high time complexity), ability to capture nonlinear dependencies (which linear and logistic regression cannot do), smoothness of predictions (which random forest cannot do given its step-like boundary), predictions for outliers (which range from simple extrapolation to a constant 50-50 prediction), as well as interpretability. These features vary both by ML algorithm and by the chosen hyperparameters, which is why the interactive plots help visualize such limitations. For example, the linearity of the random forest decision boundary depends on the maximum depth hyperparameter (i.e. increasingly nonlinear for greater maximum depths).

These limitations can lead to bias in results. For example, outliers predicted by a random forest classification may be predicted with high predicted probability despite being far from the training data points. Across all these models, there is a bias-variance tradeoff whereby low complexity models underfit the data and high complexity models overfit. For example, the SVM plot shows that SVM overfits for high gamma values (low smearing around data points) and underfits for very low gamma values (high smearing around data points) resulting in high bias and a linear decision boundary incapable of capturing nonlinear patterns. Similarly, the XGBoost plot shows that XGBoost can overfit for low regularization values (alpha and lambda) and underfit when the regularization is too high. Thus, visualizing these trade-offs dynamically across sliders allows users to connect hyperparameter selection to bias in classification results.

In terms of data limitations, the toy dataset used only (and intentionally) shows a simple 2-feature binary classification problem. Thus, it does not illustrate classification problems with many (3+) features or target classes which are typical of real-world classification problems. The number of points and degree of noise can be toggled to test ML models' performance across conditions. However, the limitations of this dataset limit the analysis to binary classification problems.

### 4. Understand Societal Impacts of Data and its Analysis

Visualizing ML models' classification may help users understand, appropriately apply, and explain ML models. As such, such visualization may limit users applying machine learning models as a black box, which can exacerbate bias. Every model has potential for bias from the input data, algorithm used, and the scope of its conclusions. Depending on the uses of ML, such bias (i.e. systematic errors or prejudiced results) can reinforce harmful societal biases and create high human costs (Mehrabi et al., 2021).

While ML models are often considered objective and unbiased because they are mathematical, biases in input data and limitations in the given algorithm can lead to biased results which can

reinforce discrimination. This can occur across domains such as college admissions, loan applications, targeted advertising, policing, criminal justice sentencing, surveillance, credit lending, insurance, social media, and politics (O'Neil, 2017). Across these areas, large-scale ML models that make decisions by quantifying human traits often encode existing societal biases in negative feedback loops that reinforce injustice. Part of mitigating this harmful societal impact involves more transparency around how ML algorithms work, what data is used, and what the limitations are of a particular model. Reducing blind faith in algorithms as purely objective and cultivating a better understanding of how and when models work (or don't) may help improve models' usage and limit harmful bias. Via the interactive visualization tools, this project aims to contribute to these goals.

Importantly, the project results show how sensitive the classification output is to changes in hyperparameter values, as well as to the input data. They also show the trade-offs involved in hyperparameter selection, the limits of particular models, and how the results necessarily depend on the choices (e.g. of model, hyperparameter values, data) of the particular programmers or data scientists involved. Thus, understanding these choices and trade-offs visually may help students select appropriate models and consider the potential for bias.

## Tool-chains Used

For this project, the following tools were used:

- Plotly
- Ipywidgets
- Pandas
- Scikit-learn
- Numpy
- Jupyter notebook
- Github
- Conda

For plotting the interactive plots, I used a combination of Ipywidgets (a Python library of HTML interactive widgets for Jupyter notebook) and Plotly. To create interactive plots with multiple sliders, I used Ipywidget's `interact` function, which creates UI controls for function arguments, and then calls the function with those arguments whenever you change the controls (such as a slider) interactively (Jupyter Widgets Community, 2015). I created the figures to interact with using Plotly Graph Objects - Python classes that represent figures in tree-like data structures with attribute nodes that allow for dictionary-style lookup of properties and easy updating (Plotly Technologies Inc., 2015). In particular, I used `FigureWidgets`, a class of Graph Objects made specifically for Jupyter Notebooks that generates a figure that is dynamically updatable. In

addition, the generated FigureWidget has event listeners for clicks, hovering, selecting points, and zooming, allowing for many types of interaction.

For data generation and machine learning, I used Scikit-learn. For example, I used Scikit-learn to generate the classification dataset as well as to create and train the classification models.

For data processing, I used the Pandas and Numpy libraries.

For displaying and documenting the generated plots, I used a Jupyter notebook.

For hosting my code and version control I used Github. My project code is hosted in the following GitHub repository: <https://github.com/BrownDSI/data1030-visuals>.

To ensure up-to-date package versions and manage my environment, I used Conda. This allows others to reproduce my project environment in order to smoothly run the interactive plots. I created my environment using the following [YAML file](#).

## Data Description

For the interactive classification plots, I used a toy binary classification dataset to illustrate classification tasks. Given the 2D interactive plot, the toy dataset has 2 features and the color of the points represent the true classes.

In particular, I used Sklearn's make\_moons toy dataset, which makes two interleaving half circles with a given number of samples labeled by class membership (0 or 1) (Scikit, 2022). This dataset works well to visualize classifications as it contains 2 features and a non-linear data pattern. In addition, you can vary the number of data points and the degree of noise added to modify the classification task.

For the interactive gradient descent plots, I used Sklearn's make\_regression toy dataset, which generates a random regression dataset from an underlying linear model with a given number of points, degree of noise, and bias (Scikit, 2022). This dataset works well to visualize simple regression problems and to compute cost functions from the generated data. In particular, following the DATA1030 syllabus I used a regression dataset with only one feature, some noise (standard deviation of 10), and no bias (intercept of 0). As such, there is only one theta slope to estimate, as well as the intercept. This works for a 2D contour plot of the cost function where the y-axis corresponds to the slope and the x-axis to the intercept.



## Related Work and Methods

This project builds off Professor Andras Zsom's "Hands-on Data Science" DATA1030 syllabus (Zsom, 2021). In particular, the classification plots built off week 7 - supervised ML algos content's "*ml\_algos\_2.ipynb*" Jupyter notebook (which explores the random forest and SVM classifications) and the gradient descent plots from the week 6 "*linreg\_logreg*" notebook. This prior work plots static classification contour plots for several different hyperparameter combinations as well as static plots for gradient descent, all using Matplotlib. As such, this project builds on the syllabus by making these plots interactive (using sliders). Thus, the same data was used, with a similar process of data generation but generalized to a variable range of hyperparameters (for the classification plots) and modified for interactive plotting. Since interactive plots are not possible using Matplotlib, I used Ipywidgets instead.

This work also relied on Sklearn classification algorithms (scikit, 2022), such as:

- Logistic Regression
- Random Forest
- Support Vector Machine (SVM)
- XGBoost
- K-Nearest Neighbors (KNN)

Using Sklearn avoids coding the machine learning algorithms from scratch and ensures efficient implementations. Moreover, this allows the user to easily supply any of these ML algorithms as an input to the data generation and plotting functions to produce their desired classification plot.

## Contributions

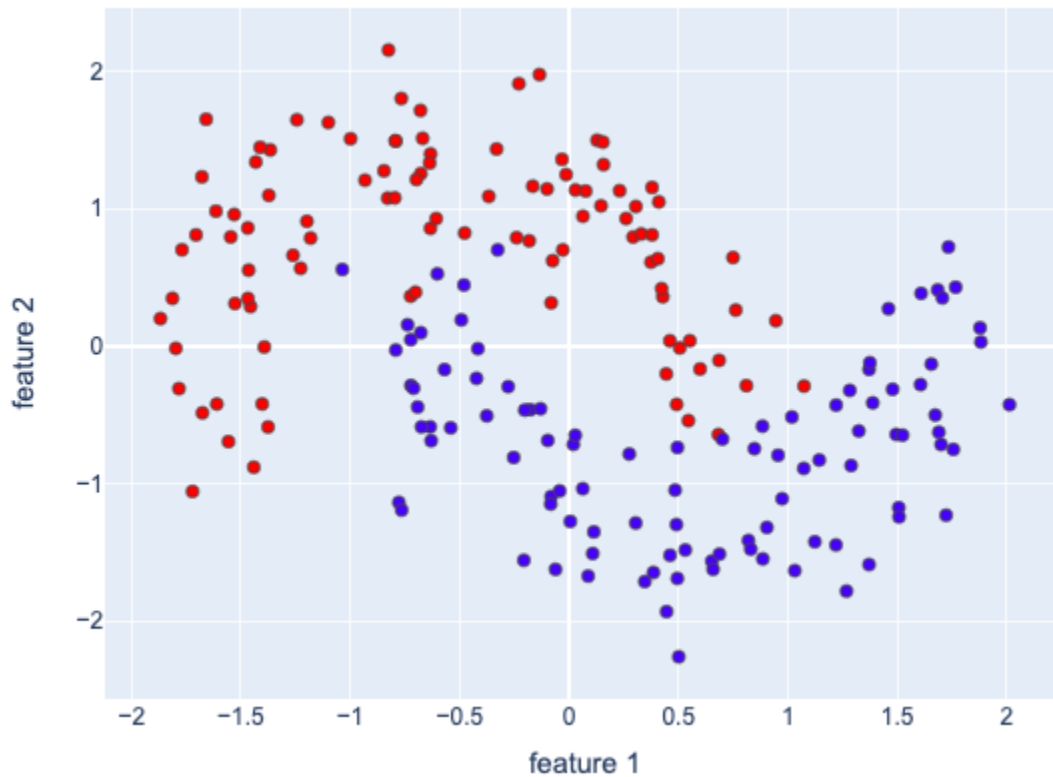
My contributions to this project were the interactive plots for classification ML algorithms as well as the interactive plots for gradient descent. This includes the functions used to generate the data for a range of hyperparameters (based on the DATA1030 syllabus) and the functions to plot the interactive visualizations. My classmate Yifei Song contributed the interactive plots for regression algorithms. The entire project is available in the shared Github repository: <https://github.com/BrownDSI/data1030-visuals>.

## EDA

Given the toy classification and regression data used, the EDA only required plotting the generated datasets as the properties and patterns of the data (e.g. the number of points, classes, linearity, noise) were already set explicitly during the data creation.

For the classification plots, I used the `make_moons` dataset with 200 samples and noise (standard deviation of Gaussian noise added) of 0.2. This generated the following data where red points correspond to a class 0 and blue points correspond to class 1:

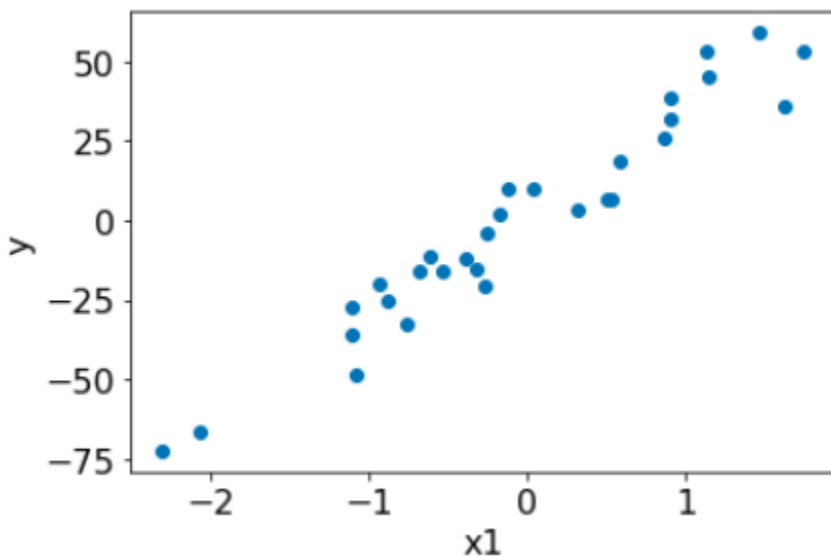
**Figure 1**  
*Classification Data*



Thus, the `make_moons` function creates two interleaving half-circles over two features. The two features, noise and nonlinearity make this data suitable for illustrating classification tasks. Some points overlap with the other class but there is a clear pattern.

For the gradient descent plots, I used Sklearn's `make_regression` dataset with only one feature, some noise (standard deviation of 10), and no bias (intercept of 0). The noise spreads the data points around the underlying linear model. Given one feature, there is only the  $\theta$  slope and intercept to estimate. This generates the following data:

**Figure 2**  
*Gradient Descent Data*



Here, as expected, there is a clear linear relationship between  $x_1$  and  $y$  with some noise. The intercept appears to approximately be at zero.

## Results

The final results are the interactive classification plots and interactive gradient descent plots, which will be added to the DATA1030 syllabus.

## Classification

The full classification Jupyter notebook can be viewed as a PDF here:

■ [classification\\_plots.pdf](#)

The classification notebook code is available on Github [here](#).

The data generation function fits a given ML classifier on given input data and calculates the predicted classification probabilities for every hyperparameter combination within a specified grid. It returns a dictionary of 2D arrays of predicted probabilities with hyperparameter values (in tuple) as keys.

The plotting function plots an interactive contour plot of predicted classification probabilities against a scatterplot of input data for a given ML classifier's hyperparameter grid, input data, dictionary of 2D predicted probability arrays, and grid step size.

The red points are class 0 and the blue points are class 1. Similarly, red areas of the contour plot indicate higher probabilities the points in that area belong to class 0 while blue areas indicate higher probabilities of belonging to class 1. The colorbar indicates how the contour plot's colors correspond to the predicted probability of belonging to class 1 (between 0 and 1). The 0.5 probability decision boundary is plotted in a bold black line.

The interactive plot uses Plotly's FigureWidgets to create a variable number of sliders to adjust each of the ML classifier's hyperparameters. As the hyperparameter values are changed, the contour plot changes to reflect the ML model's classification probabilities for the given hyperparameter combination. Thus, the plot interactively shows how different hyperparameter combinations affect each ML model's classification. In addition, the plot title automatically updates to the current hyperparameter combination values.

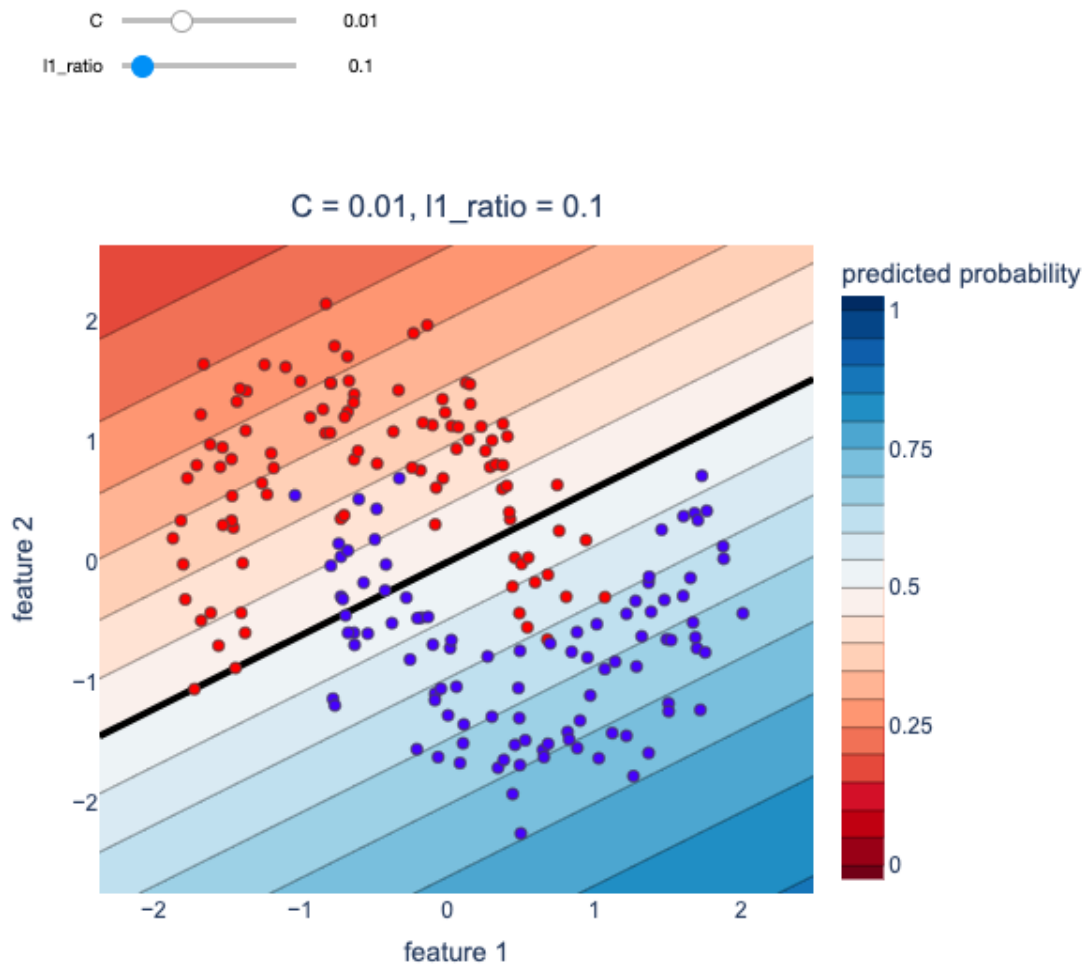
This section illustrates the resulting classification plots for the following ML algorithms for example hyperparameter ranges:

- Logistic regression
- Random Forest
- SVM
- XGBoost
- KNN

## Logistic regression

### **Figure 3**

*Logistic Regression Interactive Plot*

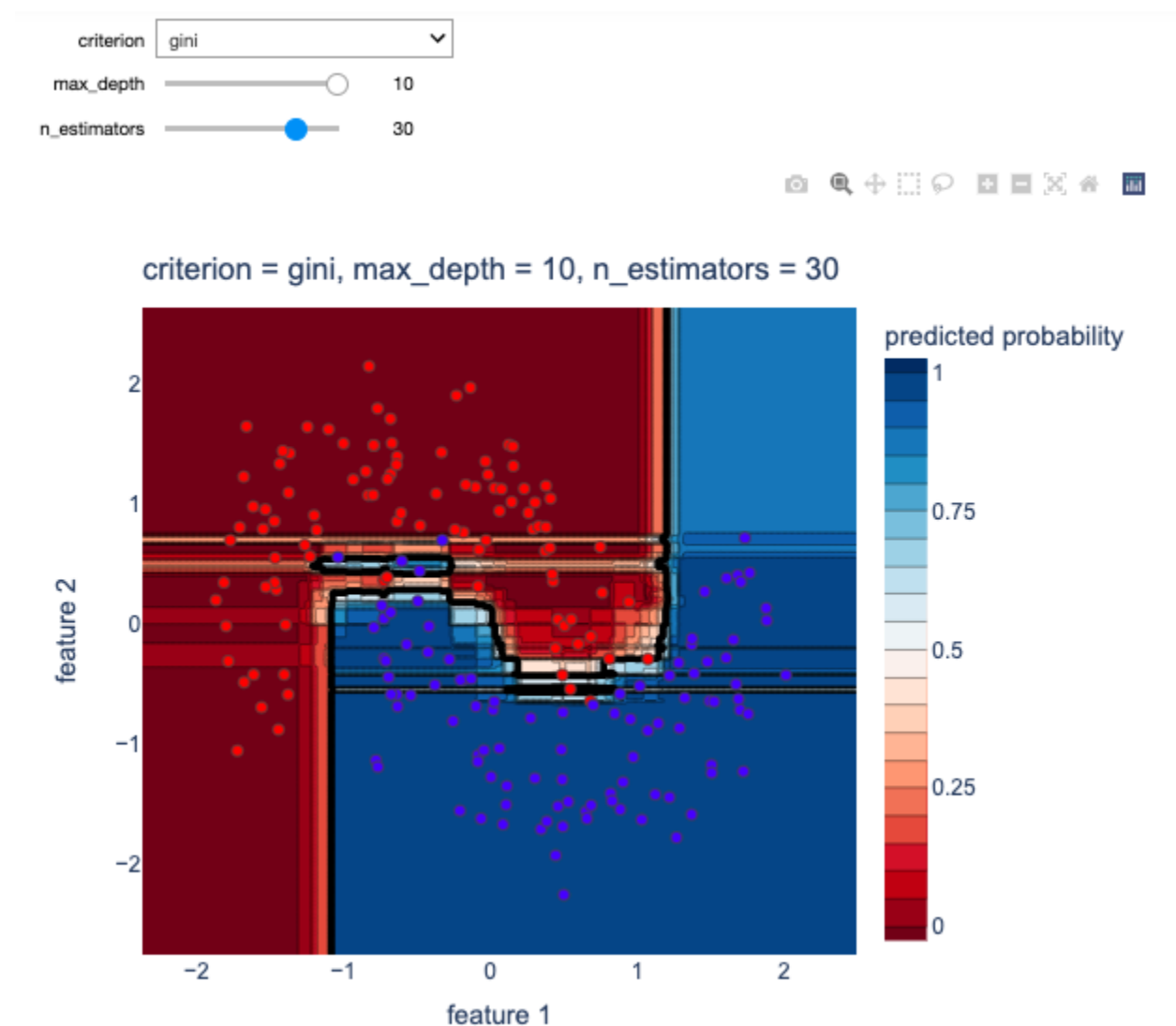


This plot illustrates how the logistic regression classification varies as a function of the hyperparameters  $C$  - the inverse of regularization strength, and 'l1\_ratio' - the ratio of L1 (lasso) and L2 (ridge) regularization penalties. The logistic regression decision boundary is linear, since this model calculates the sigmoid of a linear combination of features to determine the binary classification probability. As such, it cannot capture the nonlinear patterns of the make\_moons dataset and misclassifies many points. In addition, the model certainty increases the farther away points get from the decision boundary line. Increasing ' $C$ ', i.e. decreasing regularization strength and thus lowering the penalty on large coefficients increases the model certainty around the decision (visually tightening the contour lines). Conversely, very low values of ' $C$ ' increase the regularization and thus bias and lower model certainty around the decision boundary. Changing the 'l1\_ratio' changes the slope of the decision boundary as the coefficients for the features change based on lasso or ridge regression. Beyond a certain threshold, the decision boundary becomes horizontal as the l1\_ratio approaches full lasso regularization which shrinks the coefficient for feature 1 to zero. This threshold also depends on the degree of regularization strength set by  $C$ . Overall, this plot shows that logistic regression is only suitable for linearly separable data, and how regularization strength and type shape the predicted probabilities and decision boundaries.

## Random Forest

**Figure 4**

*Random Forest Interactive Plot*



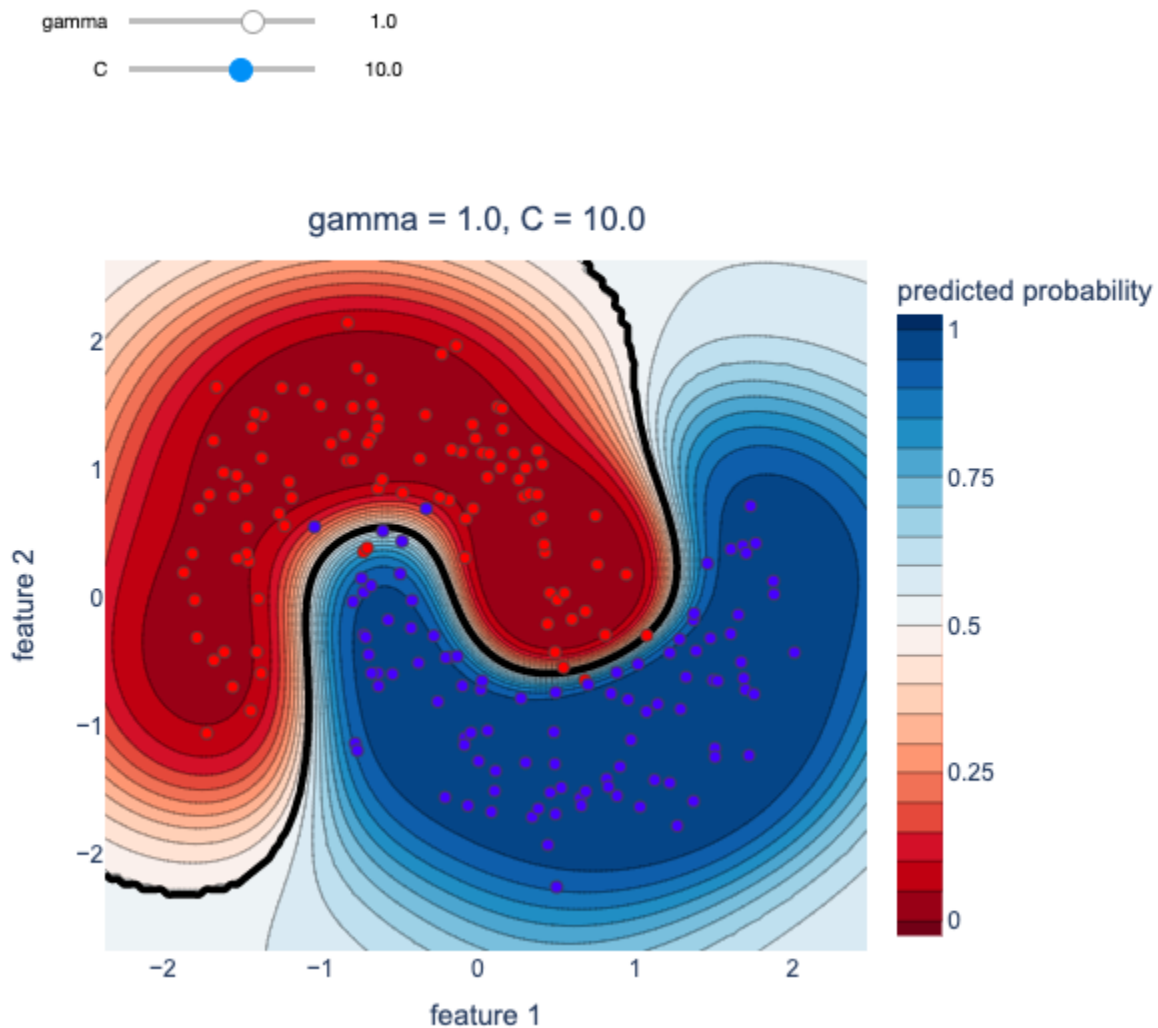
This plot illustrates how the random forest classification varies as a function of the criterion (the function to measure the quality of a split, e.g. gini or log loss), maximum depth of each tree (`max_depth`), and the number of trees (`n_estimators`). In terms of format, this plot shows 3 hyperparameters, with sliders for the continuous hyperparameters and a dropdown menu for the categorical hyperparameter. Generally, the plotting function is flexible to any number of either continuous or categorical hyperparameters (limited only by plot legibility and computing speed for generating the plot data). As the maximum depth is increased, the step-like decision boundary becomes more complex and able to capture nonlinear dependencies. Low values of maximum depth underfit the data (with a horizontal line for a `max_depth` of 1) while large values

may overfit the data (surrounding individual data points such that nearby unseen data points would be misclassified). As the number of estimators increase, the decision boundary steps become finer and fit the data points more accurately. Prediction probabilities are extrapolated by step-like sections. Thus, outliers are predicted with high probability which may be undesirable for new data points farther away from the input data. Relative to the gini criteria, the entropy and log loss criterion appear to generate slightly lower certainty predictions for outliers. Overall, intermediate values for maximum depth and number of estimators effectively classify the `make_moons` half-circles (despite step-like predictions), illustrating the importance of a good bias-variance trade-off.

## SVM

**Figure 5**

*SVM Interactive Plot*



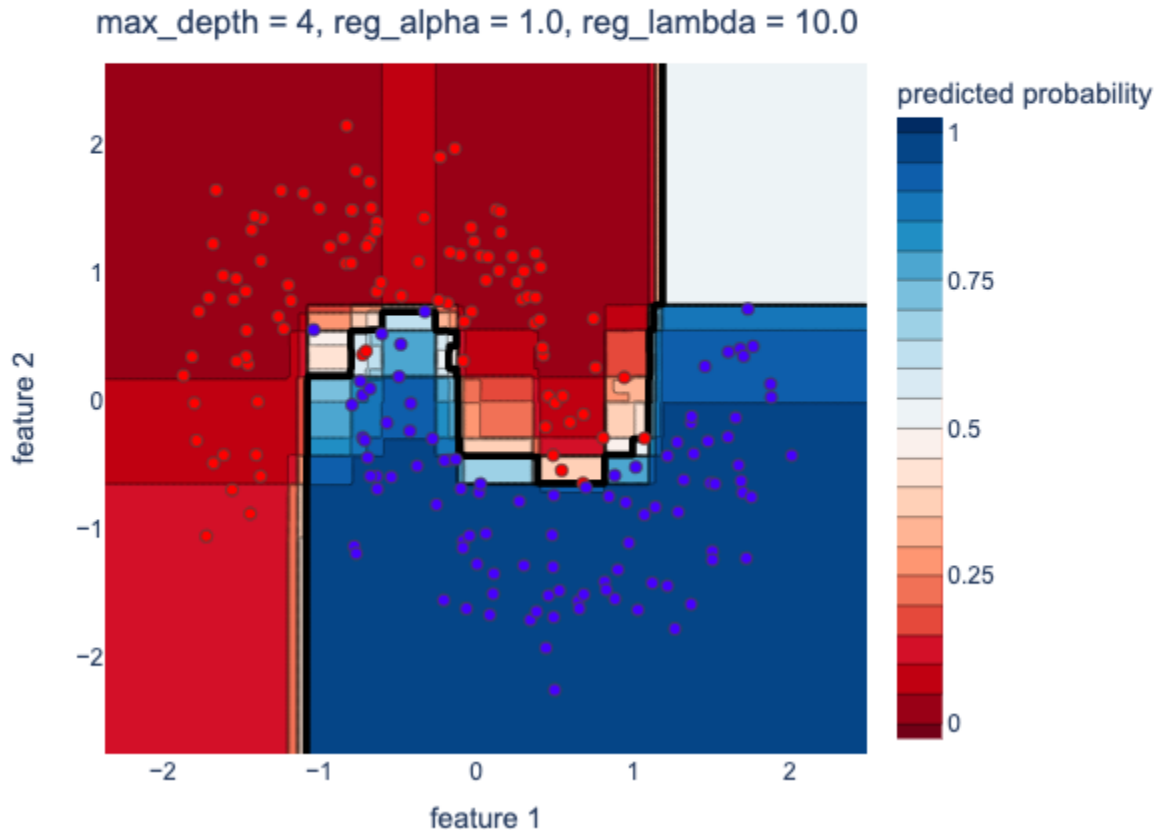
This plot illustrates how the SVM (Support Vector Machine) classification varies as a function of the gamma (the kernel coefficient, which determines the radius of influence of data points selected by the model as support vectors) and C (the inverse of regularization strength). For low values of gamma (high smearing around data points), the SVM classifier produces a linear decision boundary - i.e. high bias - and underfit the data. As gamma increases, the decision boundary becomes increasingly nonlinear. High values of gamma (low smearing) eventually overfit the data, with a decision boundary that fits to specific data points and would thus misclassify nearby unseen data points. Changing the regularization hyperparameter C changes both the shape of the decision boundary and the model certainty. As C increases (i.e. as the regularization decreases), the decision boundary becomes increasingly nonlinear and the model becomes increasingly certain. Thus, C also controls the degree of bias. Overall, the decision boundary for SVM and the predicted probabilities are nonlinear and smooth, with low certainty for outliers. This plot also shows the importance of hyperparameter selection for a good bias-variance trade-off between under and overfitting, and how hyperparameters interact to shape predictions.

## XGBoost



**Figure 6**

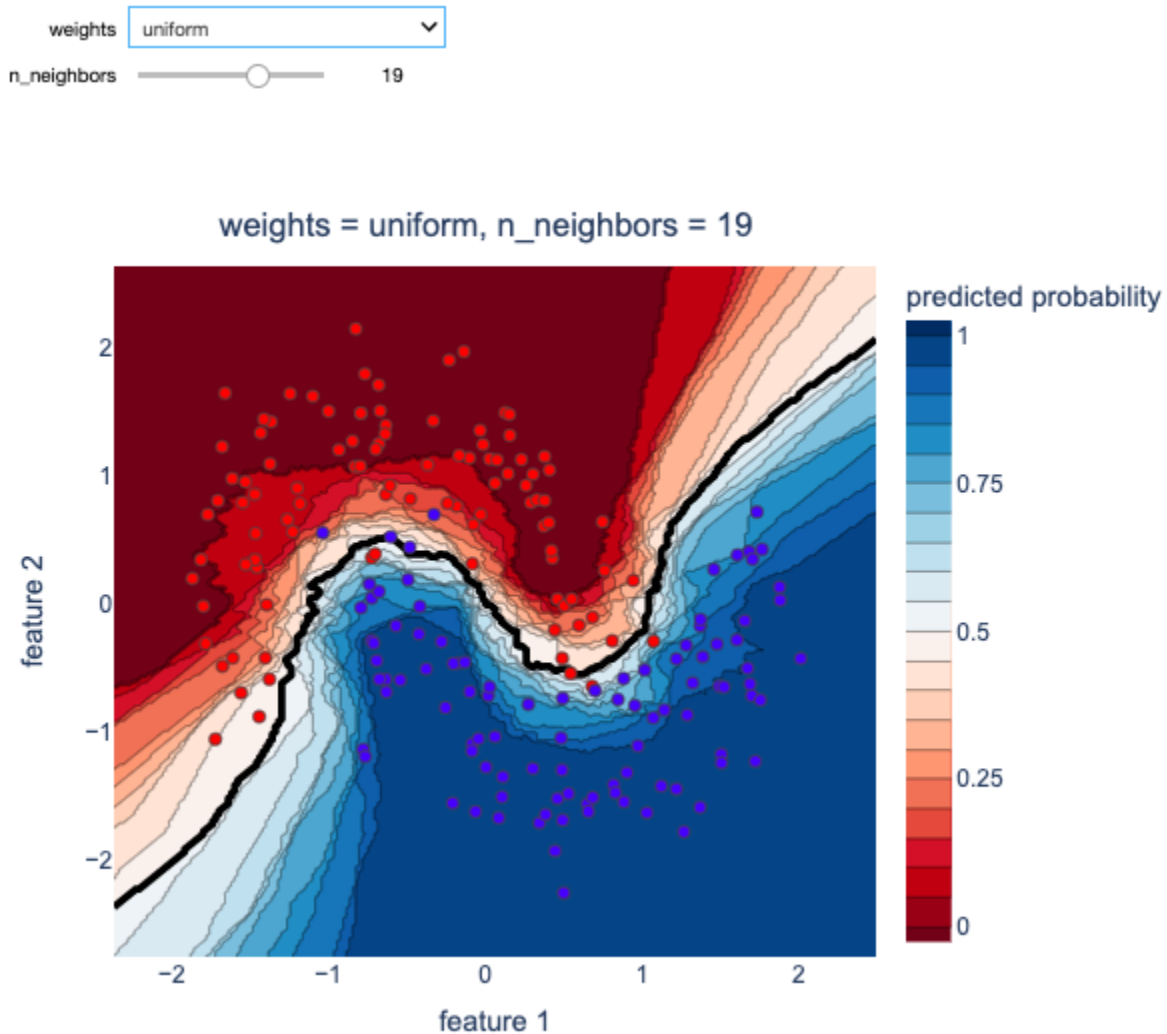
*XGBoost Interactive Plot*



This plot illustrates how the XGBoost (eXtreme Gradient Boosting) classification varies as a function of each tree's maximum depth, alpha regularization (L1 regularization on leaf weights), and lambda regularization (L2 regularization on leaf weights). When the regularization parameters are set to 0, the model behaves like the random forest, with increasing maximum depth resulting in a more complex step-like decision boundary. Increasing the regularization parameter increases the model bias and decreases the nonlinearity of the decision boundary while lowering the model confidence. While higher lambda causes leaf weights to smoothly decrease, higher alpha causes leaf weights to go to zero. Visually, alpha regularization appears to introduce more bias with more linear decision boundaries and lower model certainty compared to lambda regularization. Broadly, this plot shows how regularization shapes predictions for tree-based models like XGBoost, and illustrates the difference between lasso and ridge regression for such models.

## KNN

**Figure 7**  
*KNN Interactive Plot*




The KNN plot shows how the number of neighbors and uniform vs distance-based weights change the decision boundary and predicted probabilities. For a very low number of neighbors the difference between the weighting systems is smaller. However, for a larger number of neighbors it becomes clear that distance-based weights more closely fit the input data points - more accurately classifying the given input data but with more potential for overfitting. However, distance-weighted KNN may be less sensitive to outliers as they have less weight. Across both weighting systems, increasing the number of neighbors widens the spectrum of predicted probabilities as more neighbors (likely from both classes) are considered for each point. In other words, areas between data points of both classes have a lower classification certainty when more neighbors are considered. For an edge case of 1 neighbor, the classification plot is divided into two completely certain areas (0 or 1 predicted probability) (with a nonlinear decision

boundary). Moreover, for both weighting systems, increasing the number of neighbors partially flattens the decision boundary between the two classes as it becomes less sensitive to nearby points (although more so for the uniform weighting since nearby points are not prioritized).

## Gradient Descent

The full gradient descent Jupyter notebook can be viewed as a PDF here:

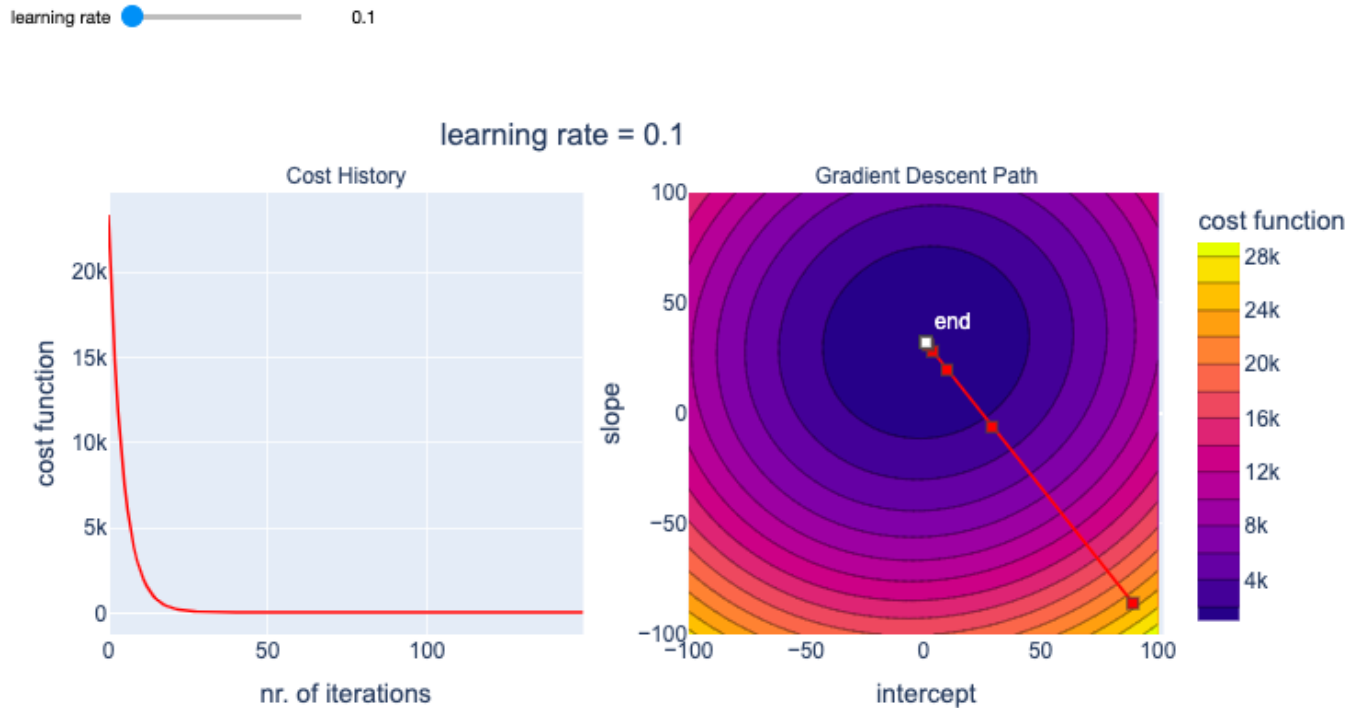
 [gradient\\_descent.pdf](#)

The gradient descent notebook code is available on Github [here](#).

The gradient descent data generation function calculates the theta (slope and intercept values) and cost history over a range of learning rates for given input data, initial (random) thetas, and learning rates. It returns a dictionary of theta history, a dictionary of cost history, and final thetas with learning rate values as keys.

The plotting code creates two side-by-side plots controlled by the same slider for learning rate. The first plot is a line chart of cost history over the gradient descent iterations, and the second plot illustrates the gradient descent path over a contour plot of the cost function. For the contour plot, the x-axis represents the regression intercept ( $\theta_0$ ) and the y-axis represents the regression slope ( $\theta_1$ ). The gradient descent path is shown in red, and the endpoint is labeled “end” in white. Thus, the slider controls the learning rate for both plots, interactively updating the cost history and gradient descent path, as well as the title.

**Figure 8**  
*Gradient Descent Path and Cost History Interactive Plot*



As the learning rate increases towards 1, the cost history converges faster (over fewer iterations) and the gradient descent path shortens. Beyond a learning rate of 1, the cost history converges more slowly and the gradient descent path crosses the minimum. For this cost function, once the learning rate exceeds 1.9 the cost history increases exponentially as the gradient descent path overshoots and diverges from the minimum. Thus, these interactive plots show how the gradient descent path and cost vary as a function of learning rate, and for which values of learning the algorithm fails to converge.

## Conclusion

This practicum project involved the creation of interactive visualizations for machine learning classification and gradient descent. These tools will supplement the DATA1030 curriculum and aim to help students intuitively understand how different ML classification algorithms work, how different hyperparameters shape predictions, the role of hyperparameters in bias-variance trade-offs, and how gradient descent operates as a function of learning rates. Through these customizable interactive visualizations, students can flexibly explore classification algorithms as well as gradient descent.

## References

- Jupyter Widgets Community (2015). ipywidgets, a GitHub repository. Retrieved from <https://github.com/jupyter-widgets/ipywidgets>
- Jupyter widgets. Jupyter Widgets - Jupyter Widgets 8.0.1 documentation. (n.d.). Retrieved August 21, 2022, from <https://ipywidgets.readthedocs.io/en/latest/>
- O'Neil, C. (2017). Weapons of math destruction. Penguin Books.
- Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6), 1-35.
- Plotly Technologies Inc. Collaborative data science. Montréal, QC, 2015. Retrieved from <https://plot.ly>.
- Plotly. Plotly Figurewidget Overview in Python. (n.d.). Retrieved August 20, 2022, from <https://plotly.com/python/figurewidget/>
- Plotly. Graph objects in Python. (n.d.). Retrieved August 21, 2022, from <https://plotly.com/python/graph-objects/>
- Scikit-Learn (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct), 2825–2830.
- Sklearn.datasets.make\_moons. scikit. (n.d.). Retrieved August 2, 2022, from [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make\\_moons.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html)
- Sklearn.datasets.make\_regression. scikit. (n.d.). Retrieved August 20, 2022, from [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make\\_regression.html](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html)
- Sklearn.ensemble.randomforestclassifier. scikit. (n.d.). Retrieved August 28, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- Sklearn.linear\_model.logisticregression. scikit. (n.d.). Retrieved August 28, 2022, from

Andrew Solomon  
DATA2050

[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

Supervised learning. scikit. (n.d.). Retrieved August 9, 2022, from

[https://scikit-learn.org/stable/supervised\\_learning.html](https://scikit-learn.org/stable/supervised_learning.html)

XGBoost Python API reference. Python API Reference - xgboost 1.6.2 documentation. (2021).

Retrieved August 28, 2022, from

[https://xgboost.readthedocs.io/en/stable/python/python\\_api.html](https://xgboost.readthedocs.io/en/stable/python/python_api.html)

Zsom, Andras. (2021). *DATA1030 Hands-On Data Science*. Providence, RI. Data Science Initiative, Brown University.