Neural Network Analysis of Ph.D. application data

Sergazy Nurbavliyevv

The University of Utah

December 13, 2019

Data set

- 759 applications from 2016-2019.

Data set

- 759 applications from 2016-2019.
- Variables: 35 variables (including DECISION and RATING).

Data set

- 759 applications from 2016-2019.
- Variables: 35 variables (including DECISION and RATING).
- Some of the variables are Emphasis Area, Sex, Undergraduate GPA, GRE Scores, etc.

Data set

- 759 applications from 2016-2019.
- Variables: 35 variables (including DECISION and RATING).
- Some of the variables are Emphasis Area, Sex, Undergraduate GPA, GRE Scores, etc.
- Total 33 input variables: 20 numeric variables and 13 categorical variables.

Data set

- 759 applications from 2016-2019.
- Variables: 35 variables (including DECISION and RATING).
- Some of the variables are Emphasis Area, Sex, Undergraduate GPA, GRE Scores, etc.
- Total 33 input variables: 20 numeric variables and 13 categorical variables.
- Goal: To predict the Decision variable and Rating.

Missing Values

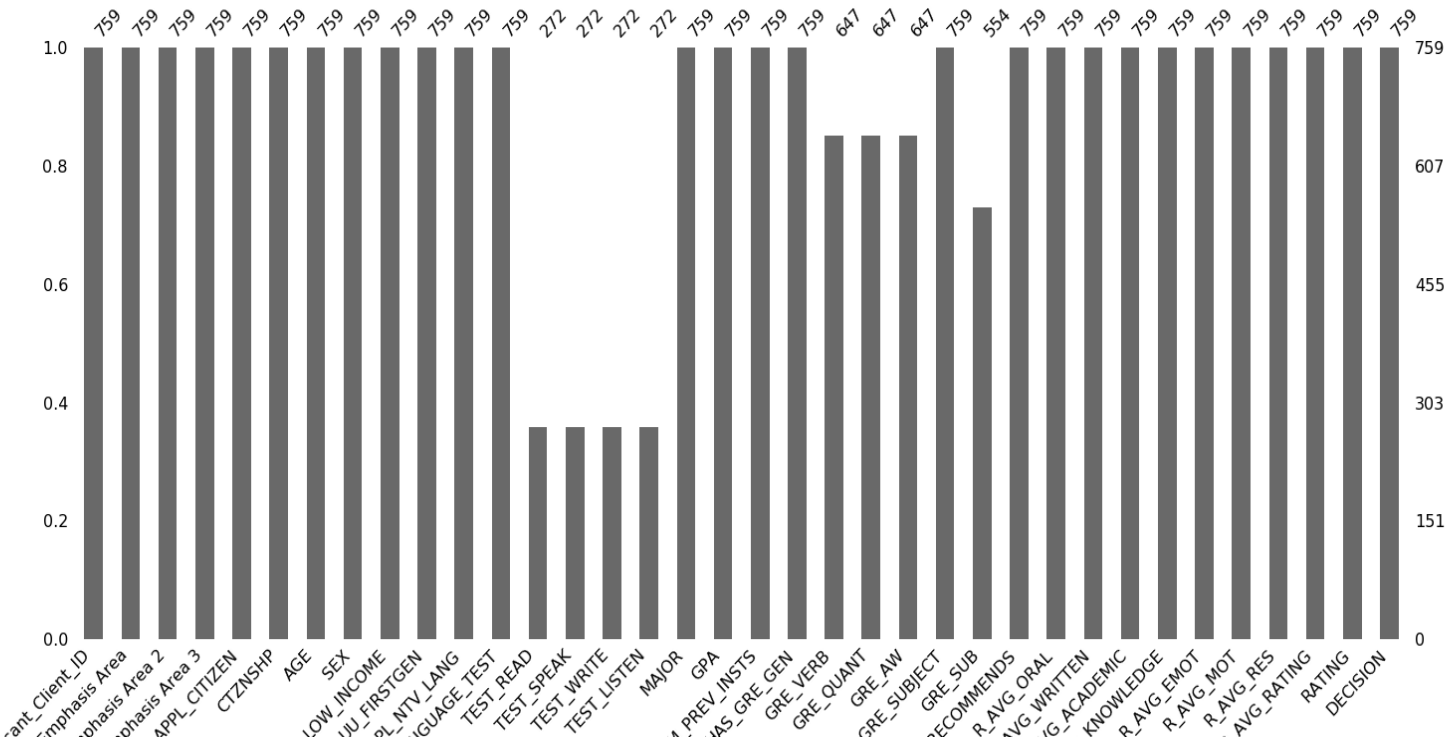


Figure: Shorter columns shows the missing values.

Data Visualization

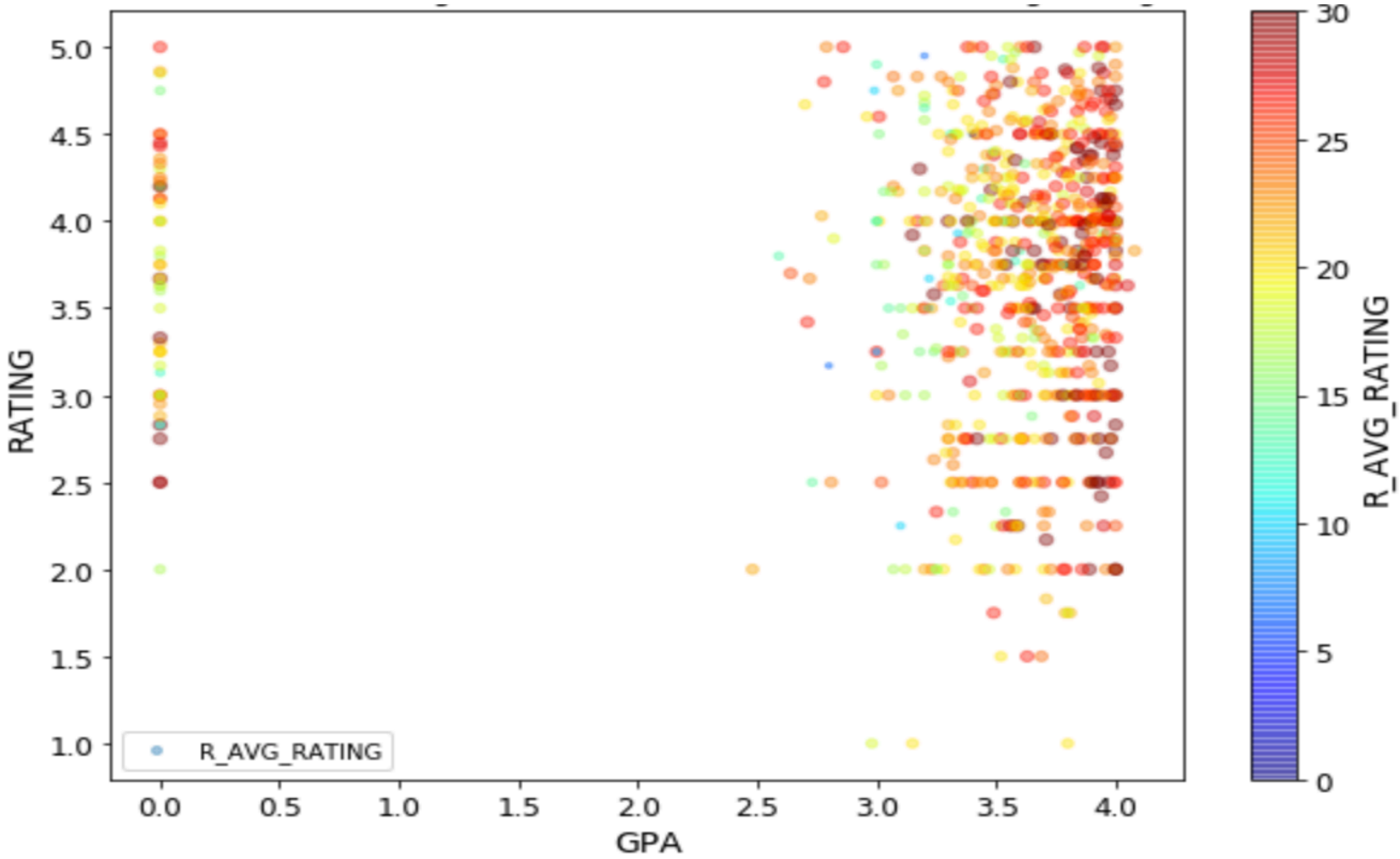
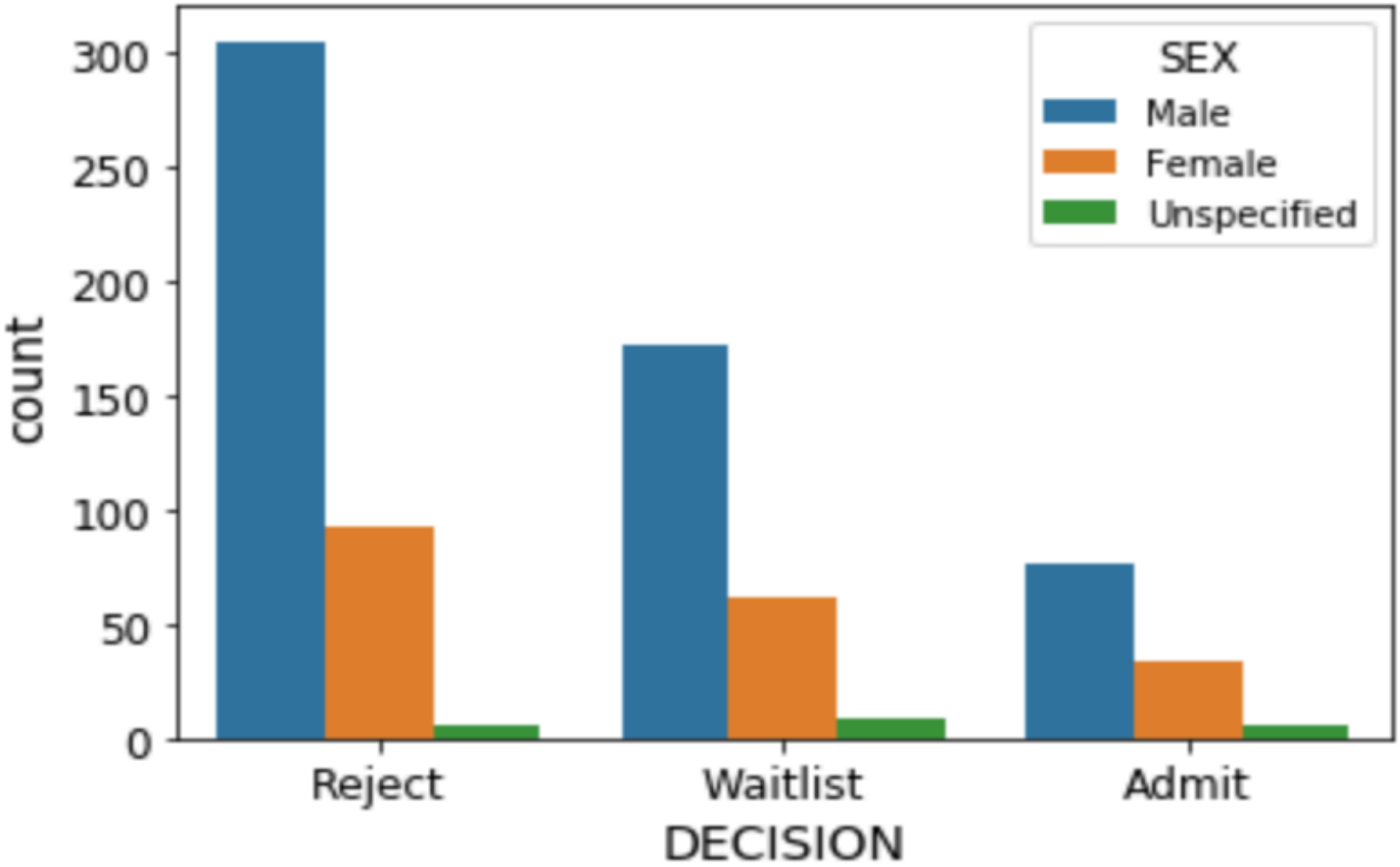
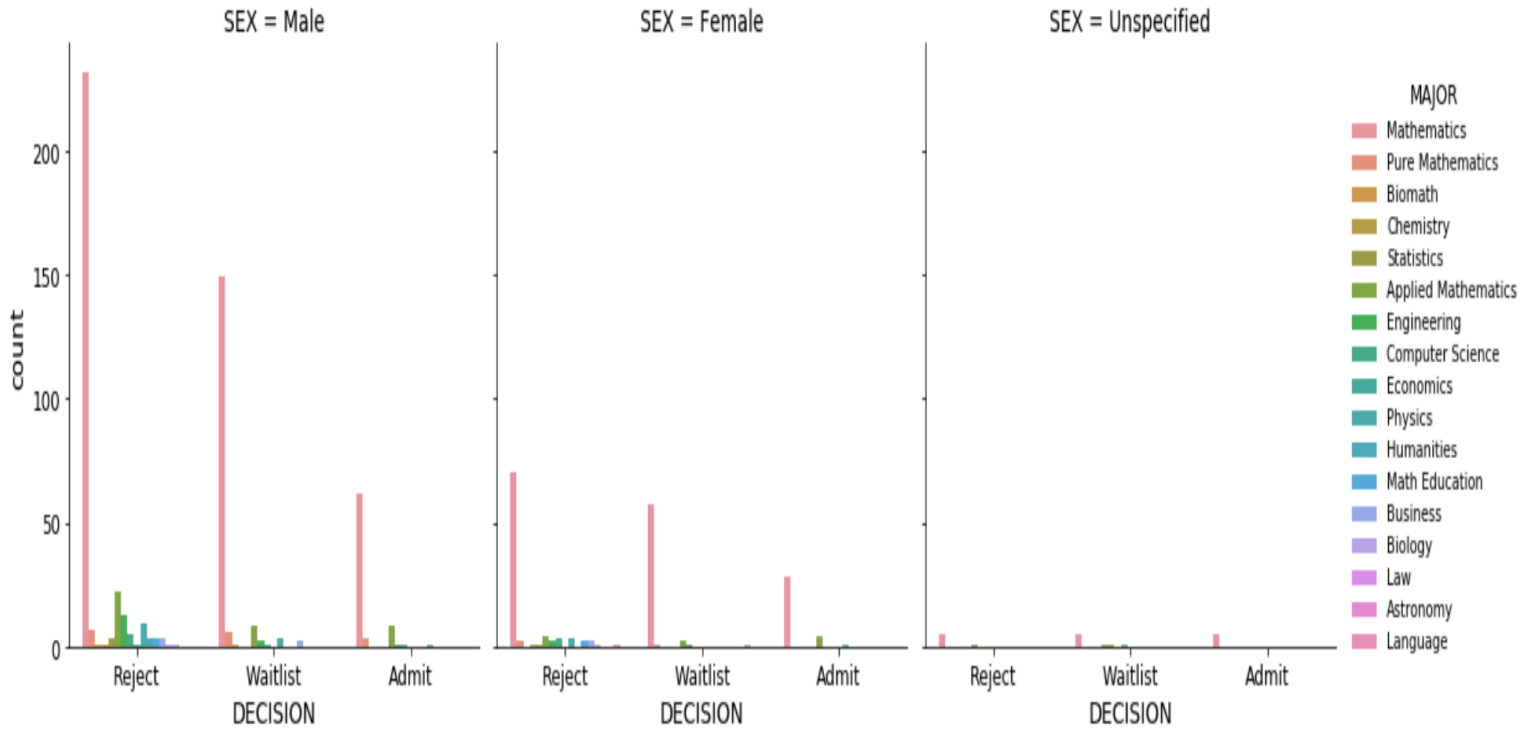


Figure: The committee rating vs GPA based on recommenders' average rating

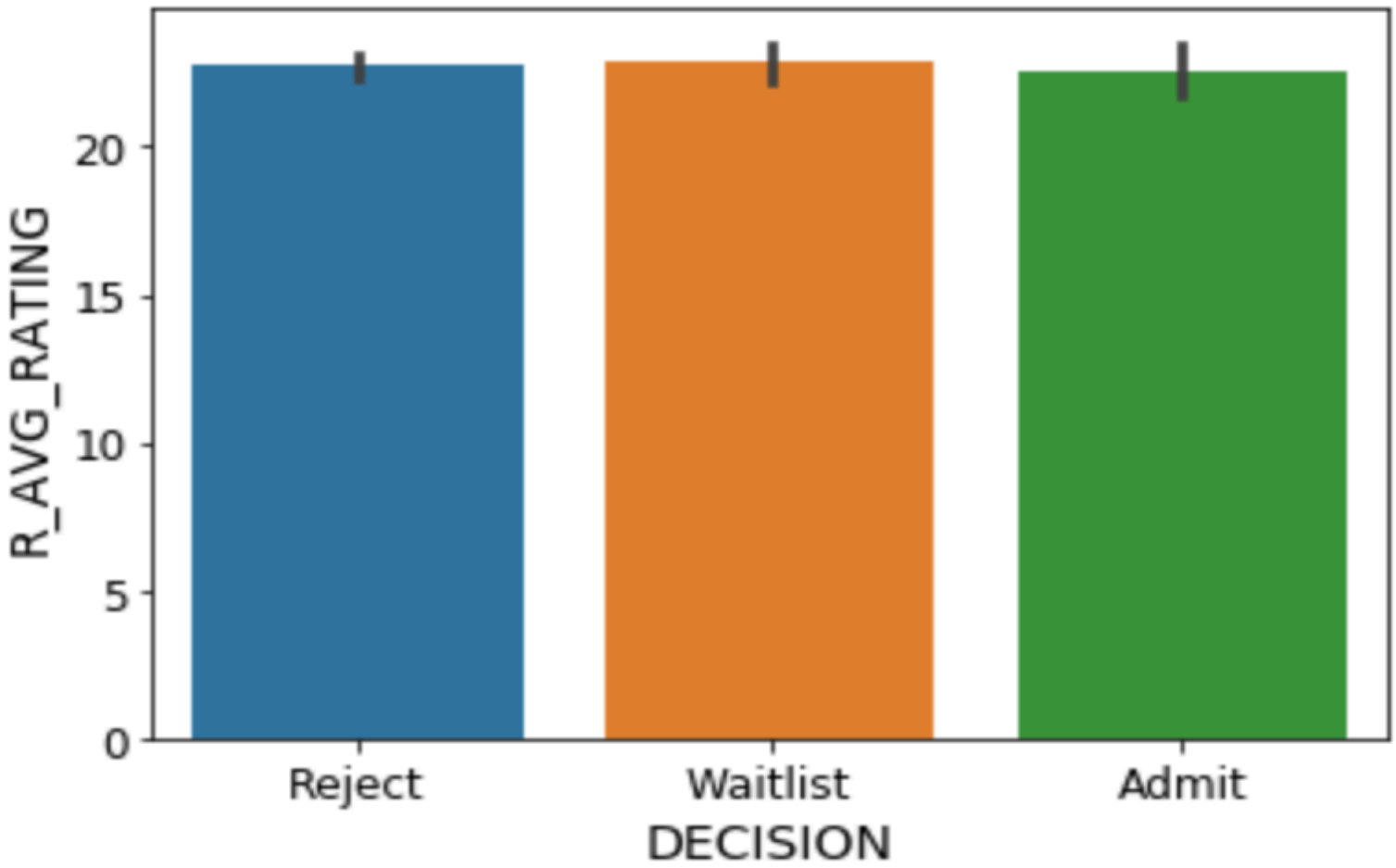
Data Visualization



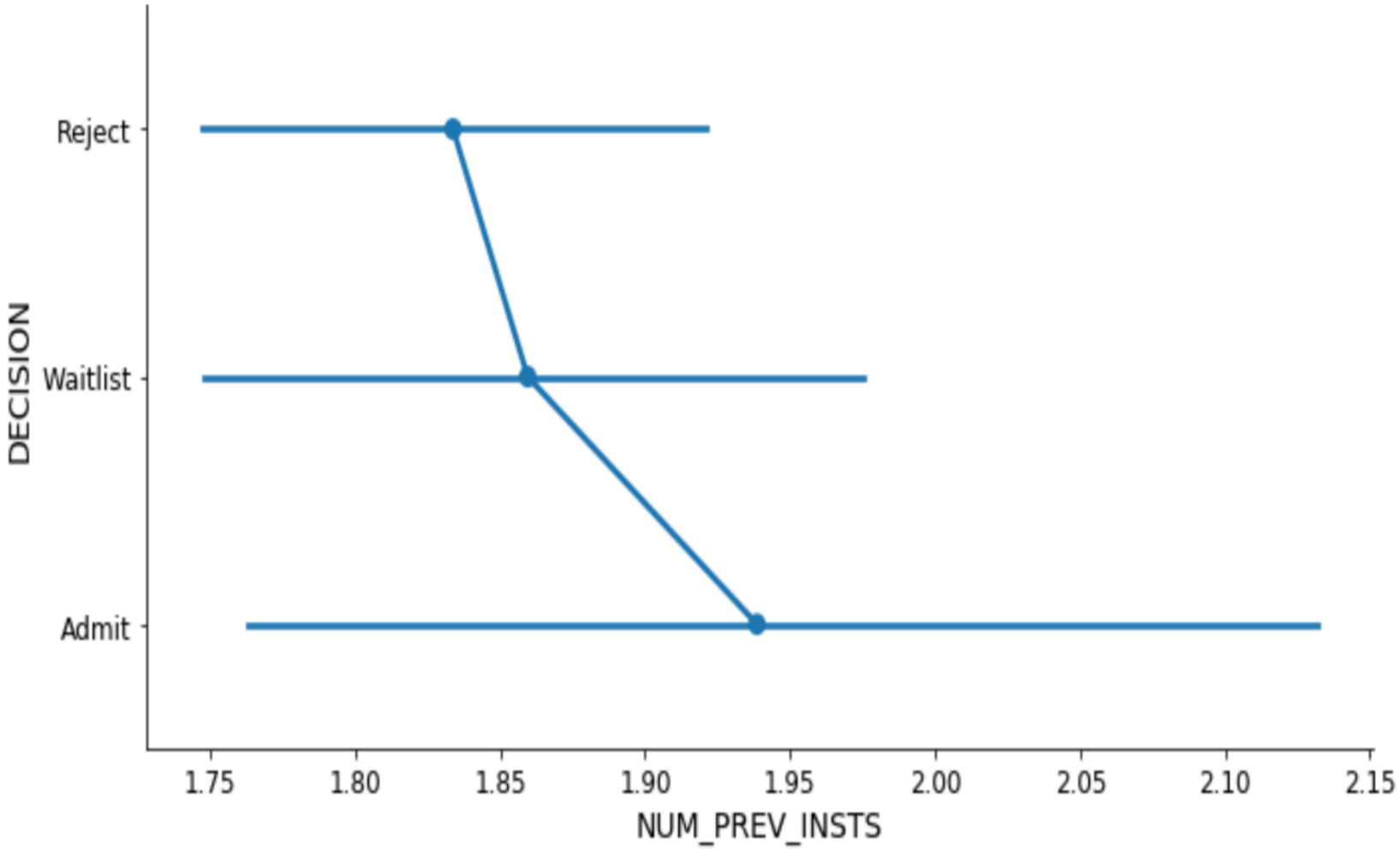
Data Visualization



Data Visualization



Data Visualization



Objectives

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent

Objectives

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent
- Neural Networks

Objectives

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent
- Neural Networks
- Stacking Approach

Gradient Descent

- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

Gradient Descent

- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.
- MSE cost function for a Linear Regression model

$$MSE(X, \theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \cdot x^{(i)} - y^{(i)} \right)^2$$

  where θ is the models parameter vector, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- x is the instances feature vector, containing x_0 to x_n, with x_0 always equal to 1.

Gradient Descent

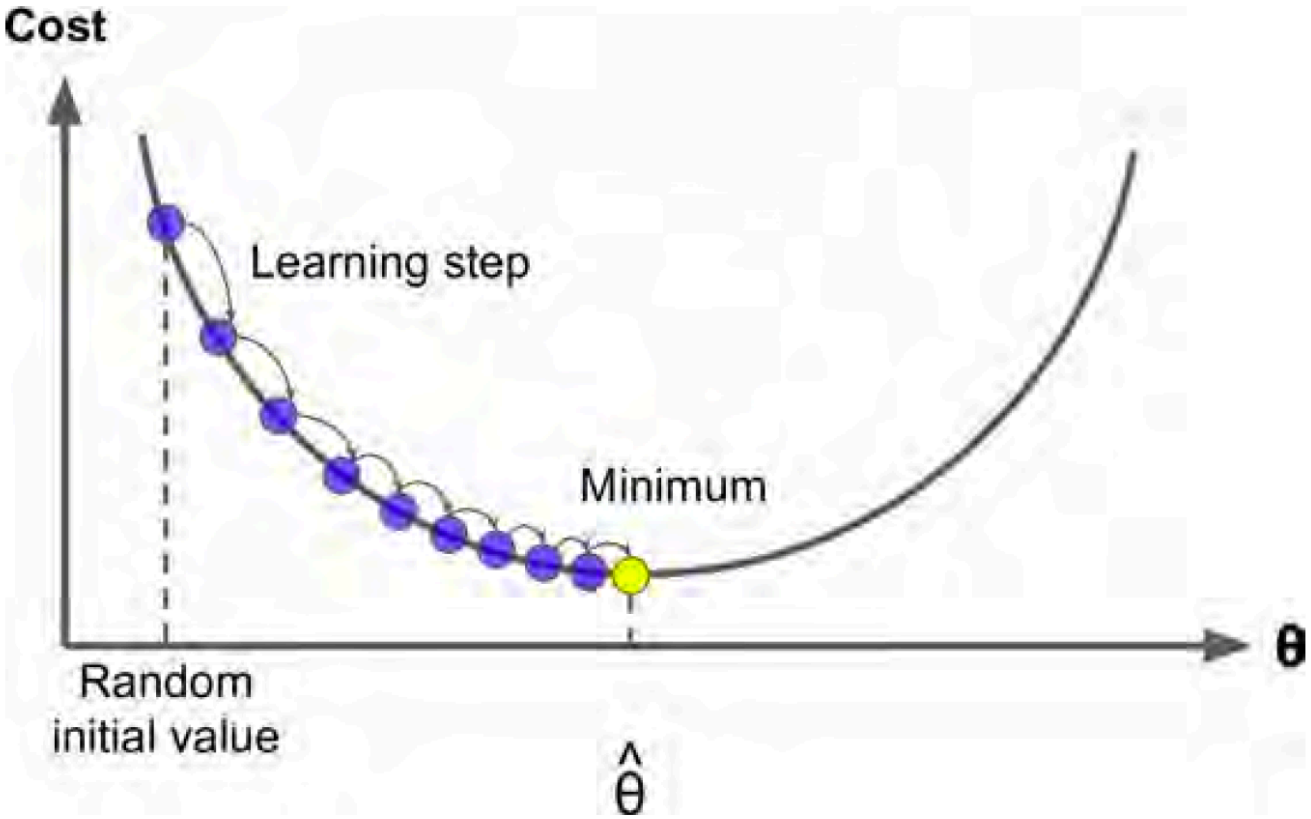


Figure: Gradient Descent

Gradient Descent

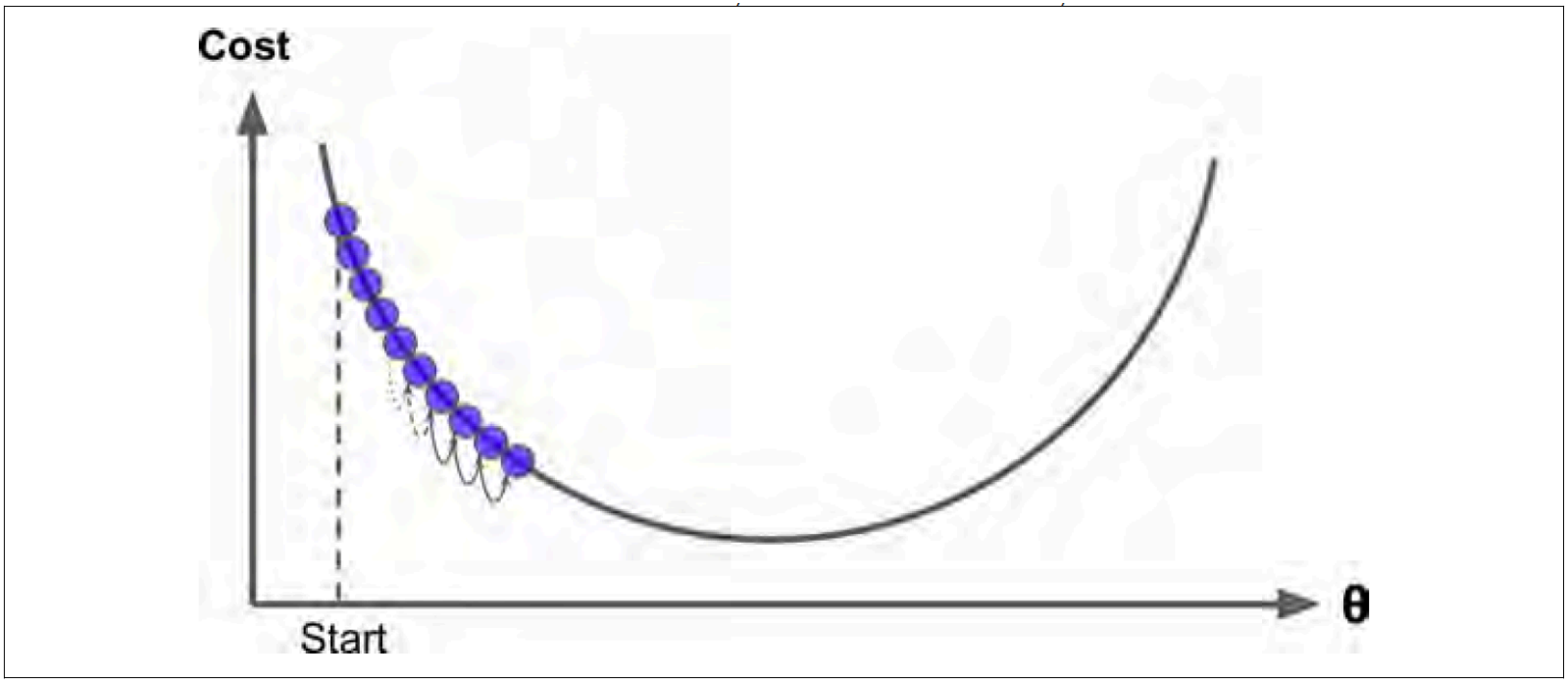


Figure: Learning rate too small

Gradient Descent

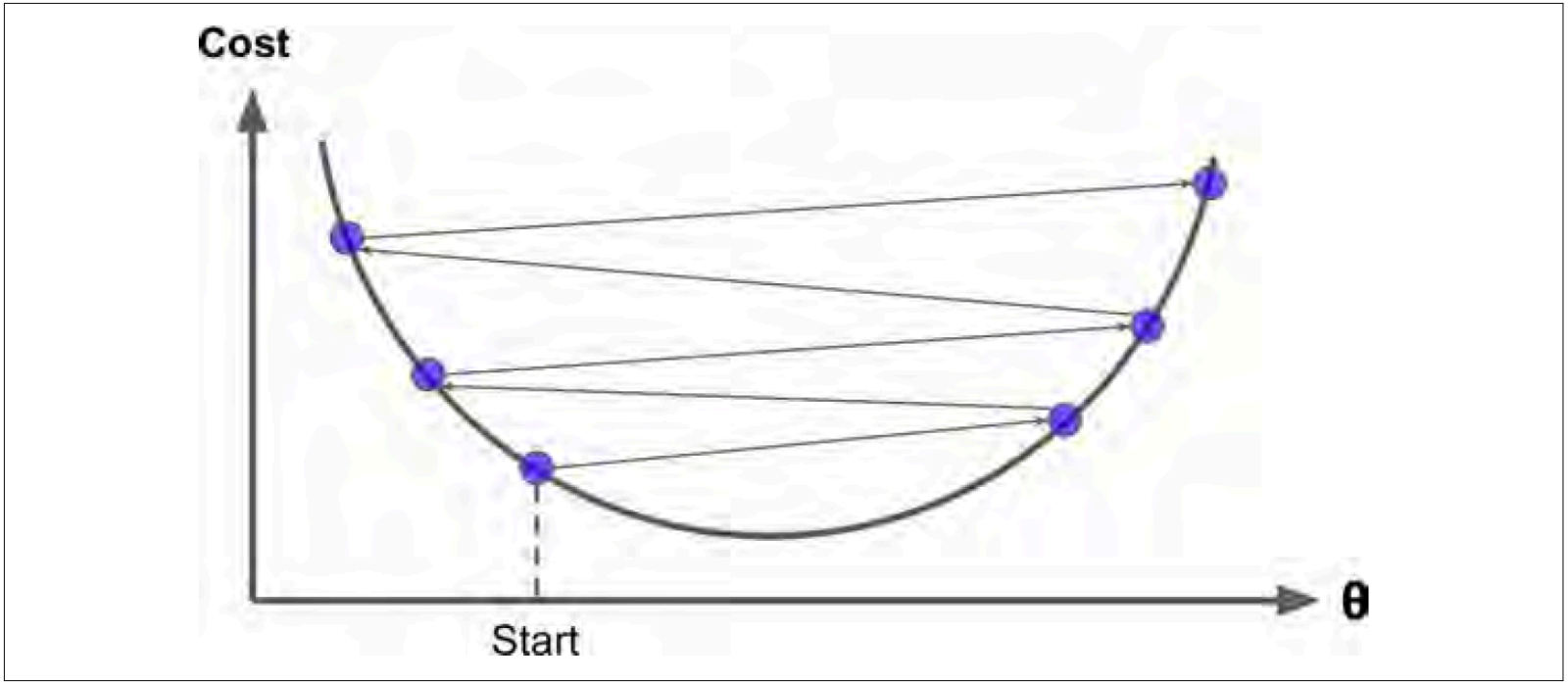


Figure: Learning rate too large

Gradient Descent

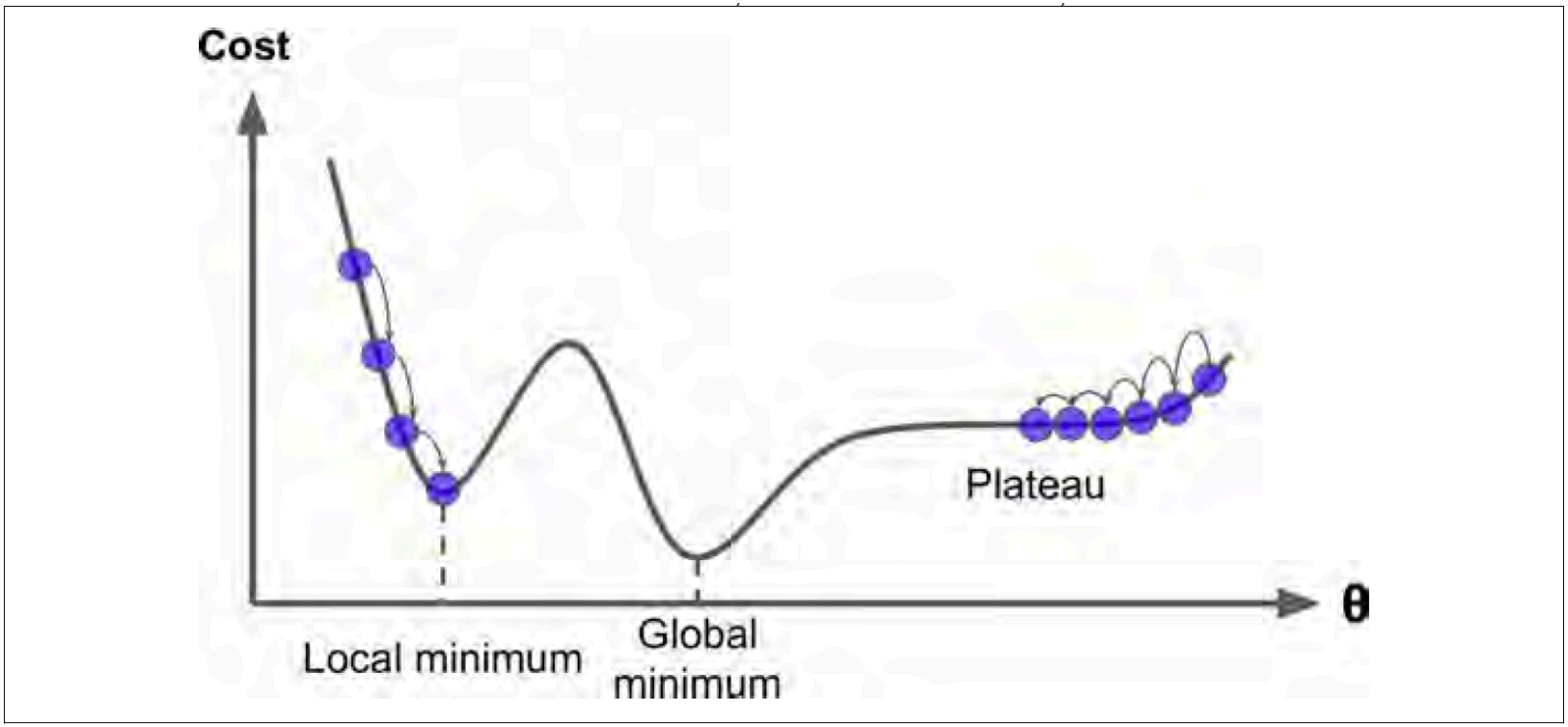


Figure: Gradient Descent pitfalls

Batch Gradient Descent

To implement Gradient Descent, we need to compute the gradient of the cost function with regards to each model parameter θ_j. In other words, we need to calculate partial derivatives

$$\frac{\partial}{\partial \theta_j} MSE(\theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T \cdot x^{(i)} - y^{(i)} \right) x_j^{(i)}.$$

Instead of computing these gradients individually, we can use

$$\nabla_\theta MSE(\theta) = \frac{2}{m} X^T \cdot (X \cdot \theta - y)$$

to compute them all in one go. The gradient vector, noted $\nabla_\theta MSE(\theta)$, contains all the partial derivatives of the cost function (one for each model parameter).

Batch Gradient Descent

Once you have the gradient vector, which points uphill, just go in the opposite direction to go downhill. This means subtracting $\nabla_\theta MSE(\theta)$ from θ. This is where the learning rate η comes into play: multiply the gradient vector by η to determine the size of the downhill step.

$$\theta^{next\ step} = \theta - \eta \nabla_\theta MSE(\theta).$$

Batch Gradient Descent

Once you have the gradient vector, which points uphill, just go in the opposite direction to go downhill. This means subtracting $\nabla_\theta MSE(\theta)$ from θ. This is where the learning rate η comes into play: multiply the gradient vector by η to determine the size of the downhill step.

$$\theta^{next\ step} = \theta - \eta \nabla_\theta MSE(\theta).$$

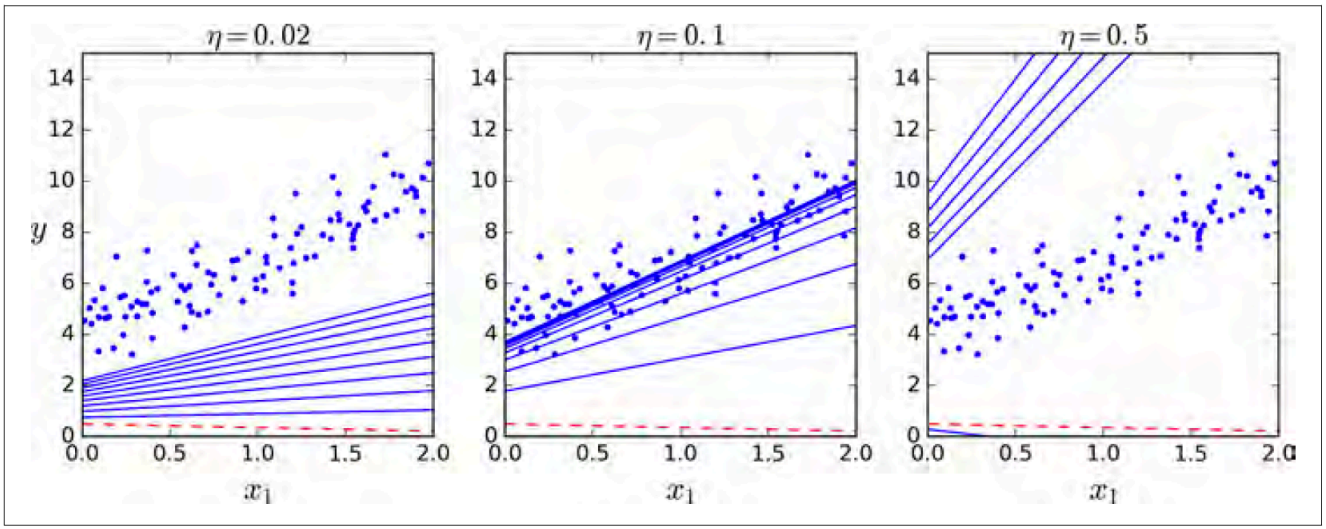


Figure: Gradient Descent with various learning rates

Stochastic Gradient Descent

- The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.

Stochastic Gradient Descent

- The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.
- At the opposite extreme, Stochastic Gradient Descent just picks a random instance in the training set at every step and computes the gradients based only on that single instance.

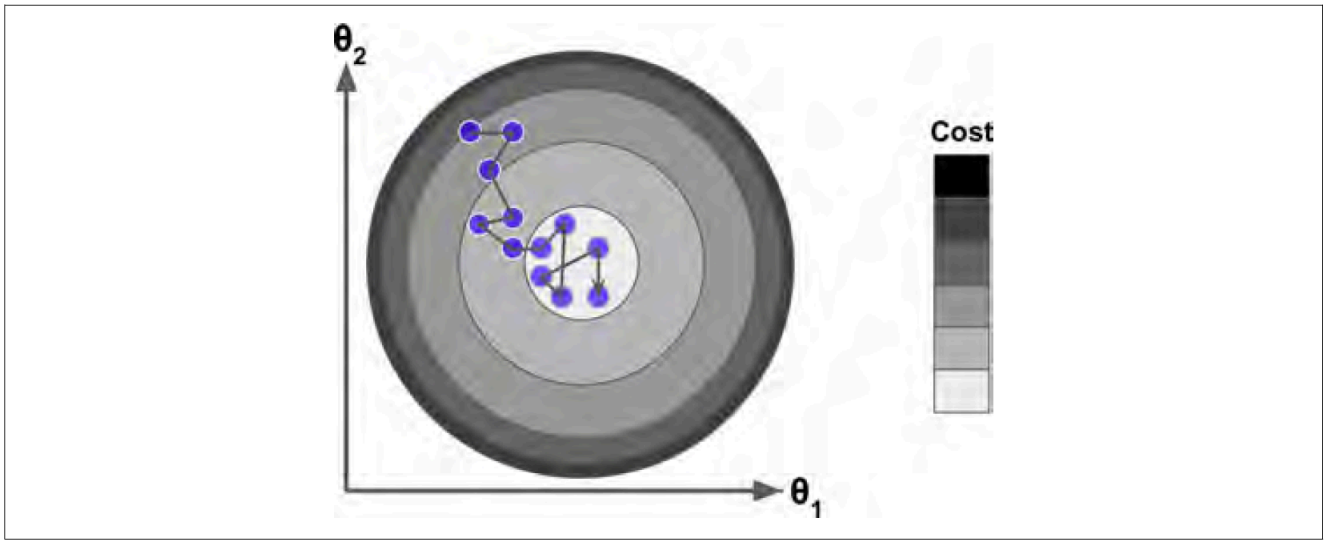


Figure: Stochastic Gradient Descent

Mini-batch Gradient Descent

- It is quite simple to understand once you know Batch and Stochastic Gradient Descent: at each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Minibatch GD computes the gradients on small random sets of instances called minibatches.

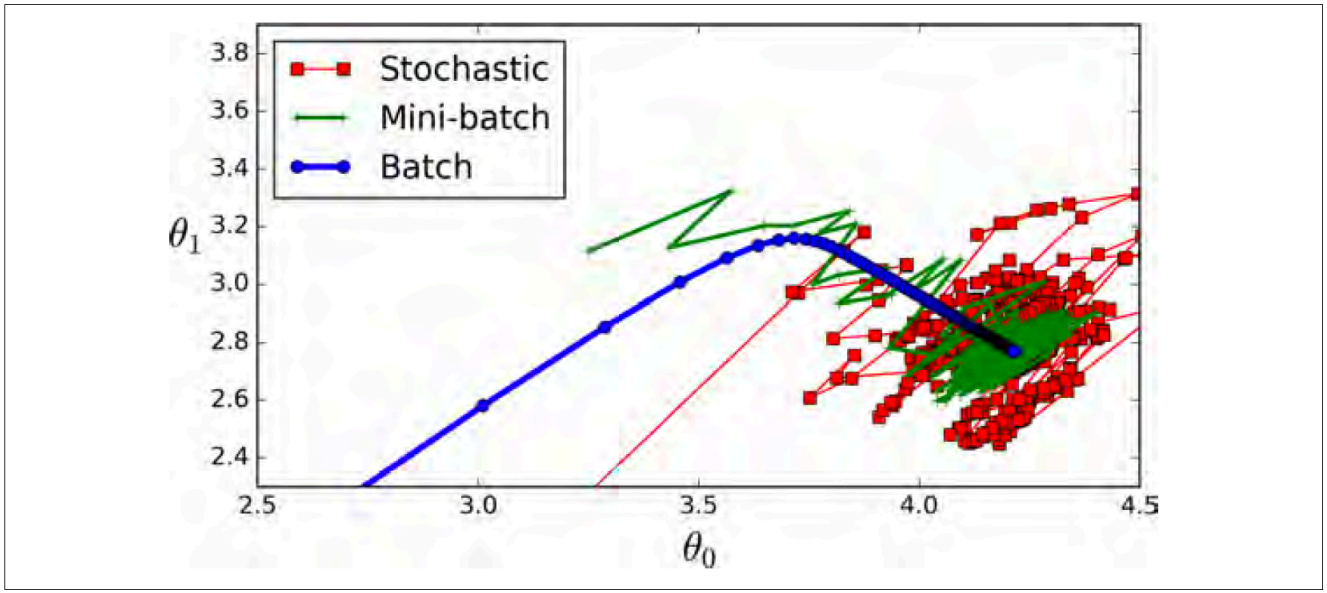


Figure: Gradient Descent paths in parameter space

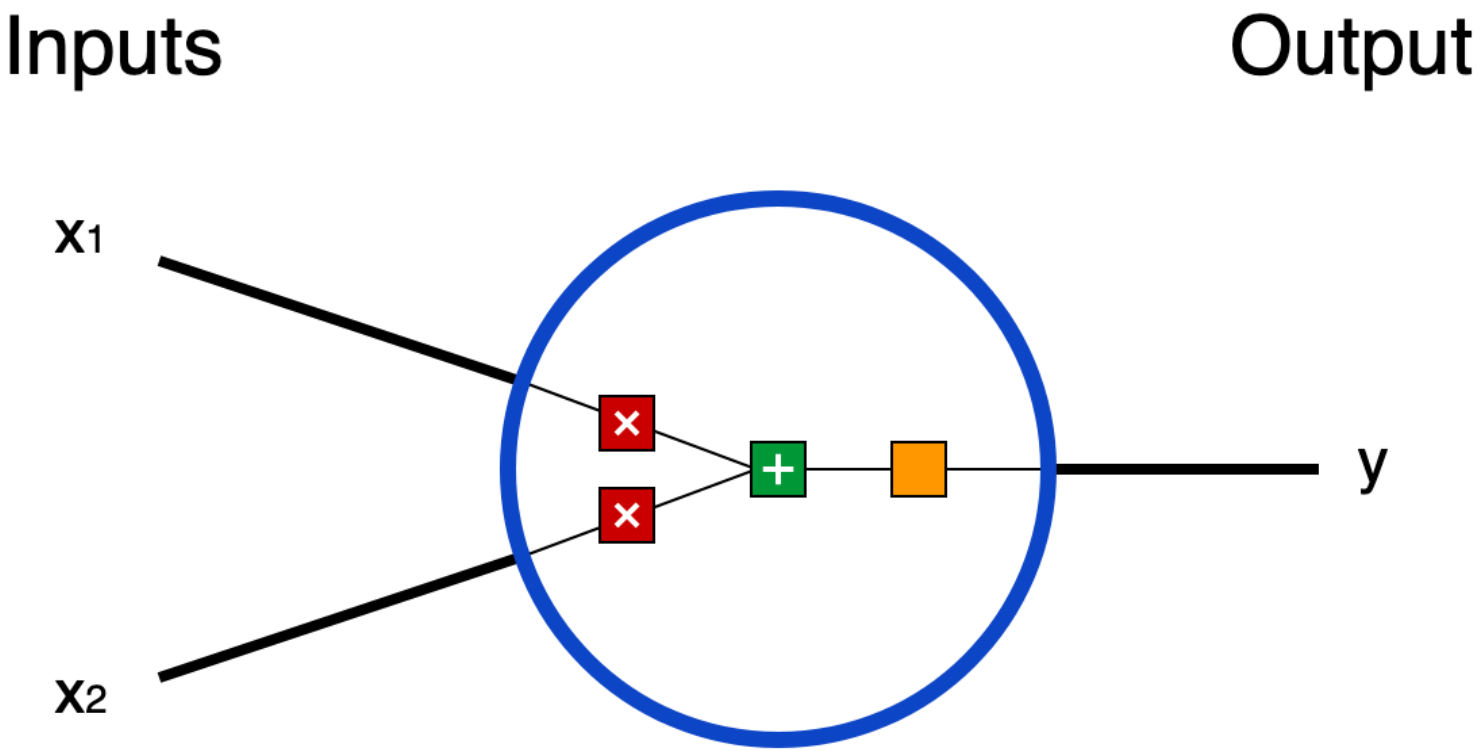


Figure: Single neuron with 2 inputs

Activation function

The activation function is used to turn an unbounded input into an output that has a nice, predictable form. A commonly used activation function is the sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$

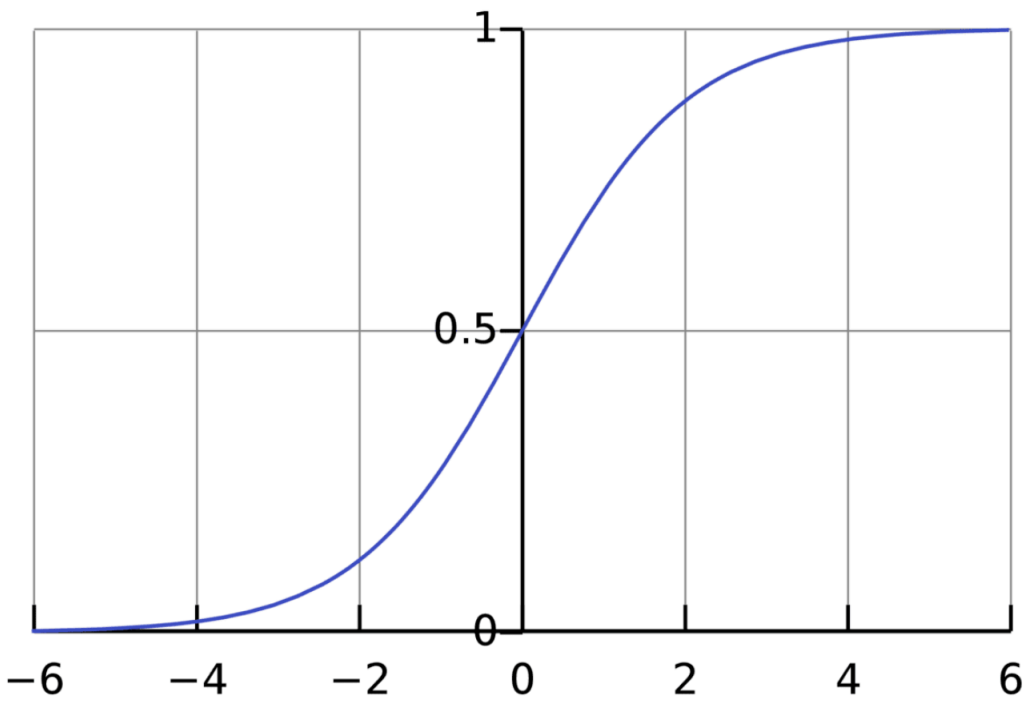


Figure: Sigmoid function

Combining Neurons into a Neural Network

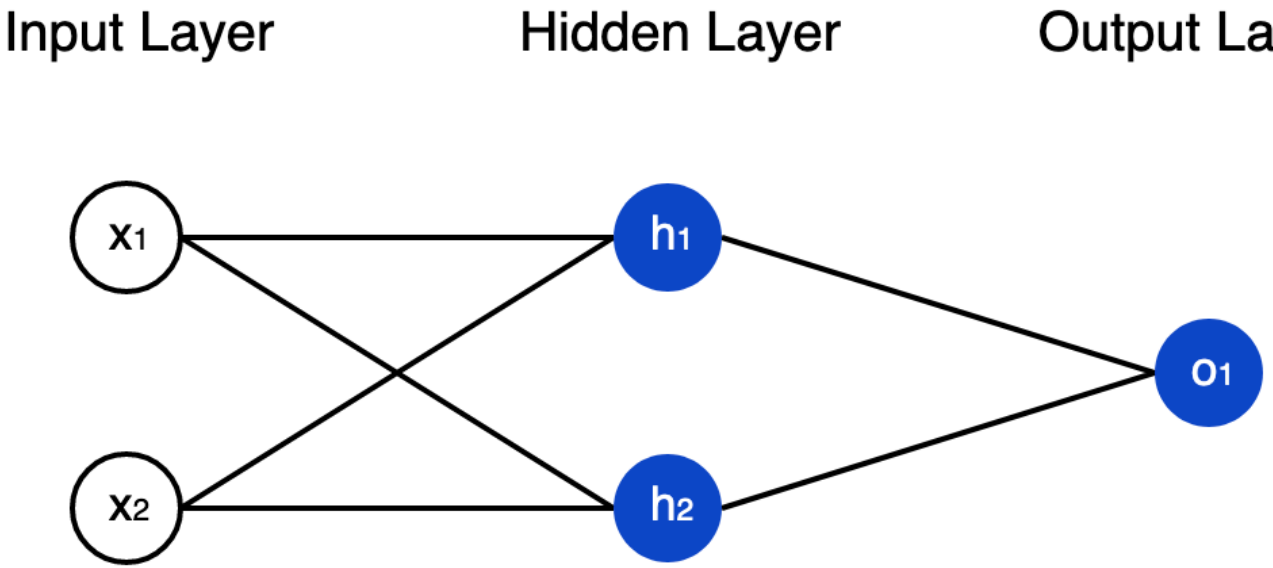
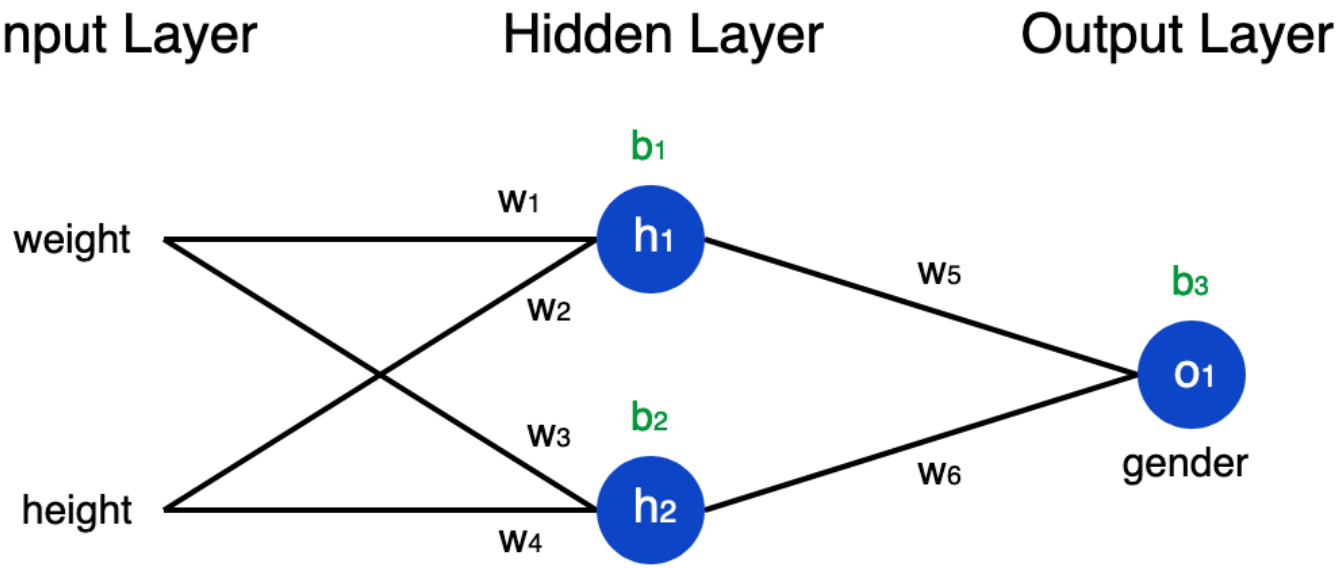


Figure: Neural Network with 2 inputs, 1 hidden layer with 2 neurons (h_1 and h_2) and 1 output neuron o_1

Training a Neural Network

- Training network to predict someones gender given their weight and height:



- Then, we can write loss as a multivariable function:

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

Back Propagation

- How would loss L change if we changed w_1? That's a question the partial derivative $\frac{\partial L}{\partial w_1}$ can answer.

Back Propagation

- How would loss L change if we changed w_1? That's a question the partial derivative $\frac{\partial L}{\partial w_1}$ can answer.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}.$$

Back Propagation

- How would loss L change if we changed w_1? That's a question the partial derivative $\frac{\partial L}{\partial w_1}$ can answer.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}.$$

- Recall $y_{pred} = o_1 = f(w_5 * h_1 + w_6 * h_2 + b_3)$.

Back Propagation

- How would loss L change if we changed w_1? That's a question the partial derivative $\frac{\partial L}{\partial w_1}$ can answer.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}.$$

- Recall $y_{pred} = o_1 = f(w_5 * h_1 + w_6 * h_2 + b_3)$.

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}.$$

- Also recall $h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$. Finally we get

Back Propagation

- How would loss L change if we changed w_1? That's a question the partial derivative $\frac{\partial L}{\partial w_1}$ can answer.

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial w_1}.$$

- Recall $y_{pred} = o_1 = f(w_5 * h_1 + w_6 * h_2 + b_3)$.

$$\frac{\partial y_{pred}}{\partial w_1} = \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}.$$

- Also recall $h_1 = f(w_1 x_1 + w_2 x_2 + b_1)$. Finally we get

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_{pred}} * \frac{\partial y_{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}.$$

Algorithm

- Update equation

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

η is a constant called the learning rate that controls how fast we train. All we're doing is subtracting $\eta \frac{\partial L}{\partial w_1}$ from w_1:

Algorithm

- Update equation

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

η is a constant called the learning rate that controls how fast we train. All we're doing is subtracting $\eta \frac{\partial L}{\partial w_1}$ from w_1:

- If $\frac{\partial L}{\partial w_1}$ is positive, w_1 will decrease, which makes L decrease.

Algorithm

- Update equation

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$

η is a constant called the learning rate that controls how fast we train. All we're doing is subtracting $\eta \frac{\partial L}{\partial w_1}$ from w_1:

- If $\frac{\partial L}{\partial w_1}$ is positive, w_1 will decrease, which makes L decrease.
- If $\frac{\partial L}{\partial w_1}$ is negative, w_1 will increase, which makes L decrease.

Stacking Approach

- Suppose we have trained a few classifiers, each one achieving about 80% accuracy.

Stacking Approach

- Suppose we have trained a few classifiers, each one achieving about 80% accuracy.
- A Logistic Regression classifier, an SVM classifier, a Random Forest classifier, a K-Nearest Neighbors classifier, and perhaps a few more

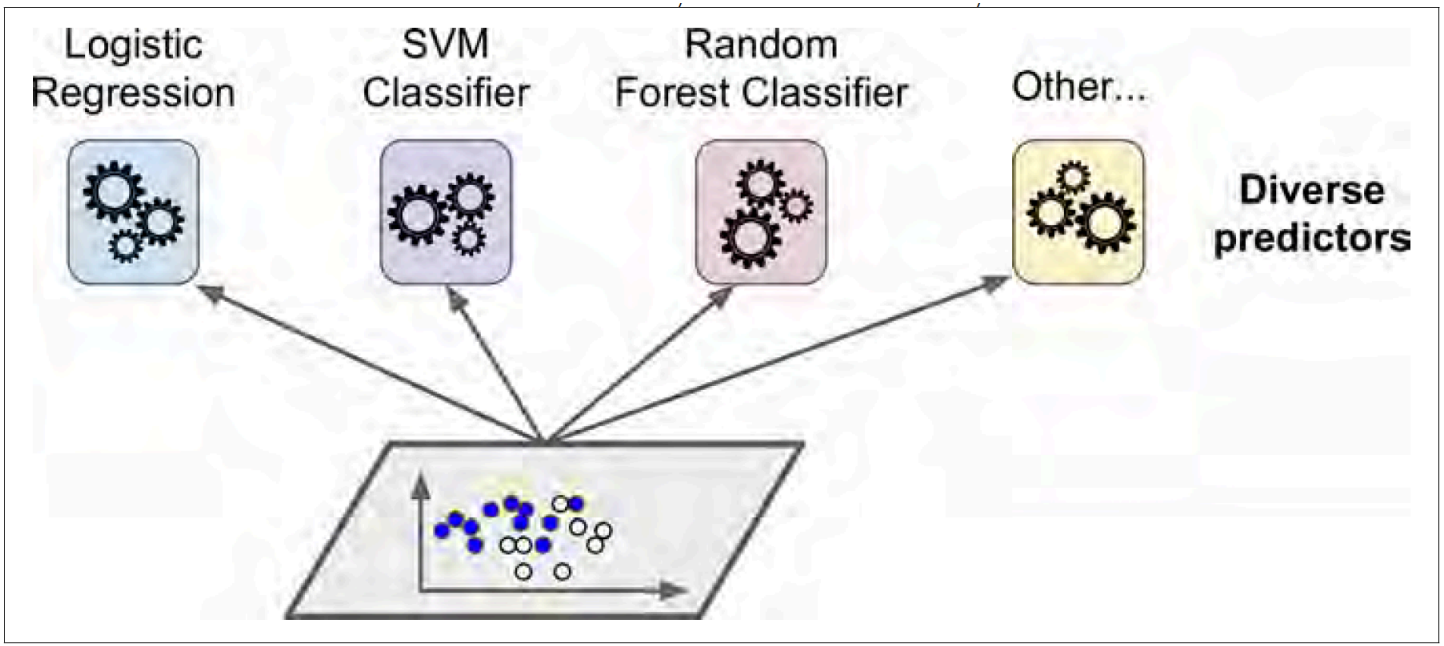


Figure: Training diverse classifiers

Stacking Approach

- A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes.

Stacking Approach

- A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes.
- This majority-vote classifier is called a **hard voting** classifier.

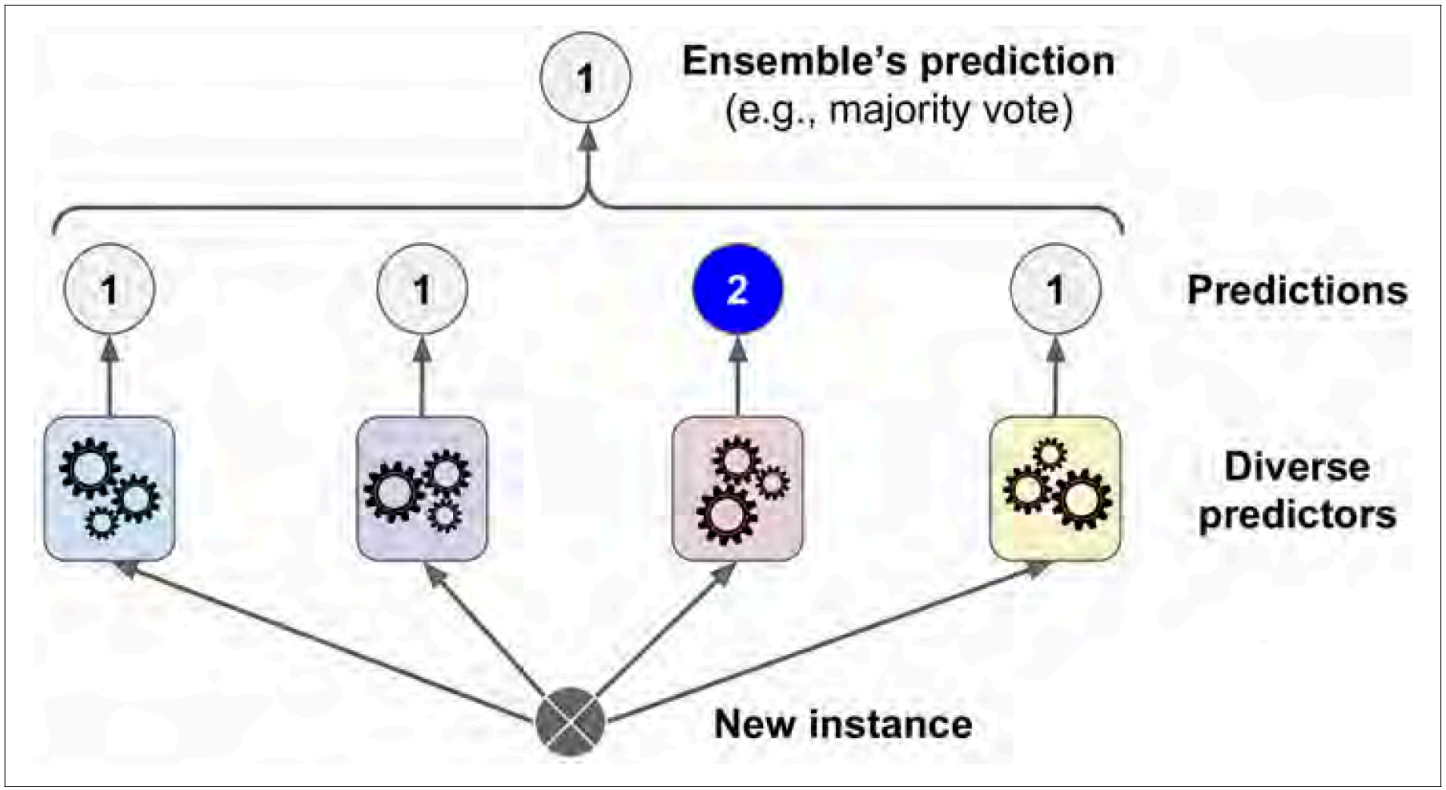


Figure: Hard voting classifier predictions

Results for predicting the Decision

Model	Score
Adaptive Boosting Classifier	0.710526
XG Boost Classifier	0.697368
Random Forest	0.631579
Stochastic Gradient Decent	0.625000
Logistic Regression	0.598684
Decision Tree	0.585526
Perceptron	0.572368
Support Vector Machines	0.565789
KNN	0.506579
Naive Bayes	0.203947

Figure: Default Parameters of the models

Model	Score
Stochastic Gradient Decent	0.743421
Adaptive Boosting Classifier	0.710526
Decision Tree	0.703947
Random Forest	0.697368
XG Boost Classifier	0.697368
Light GBM	0.684211
Linear Support Vector Machines	0.644737
Support Vector Machines	0.625000
Logistic Regression	0.592105
KNN	0.513158

Figure: Models with Grid Search

Results for predicting the Decision

Stacking approach and h2o AutoML results

- For stacking approach: average accuracy on a train set: 0.840197

Results for predicting the Decision

Stacking approach and h2o AutoML results

- For stacking approach: average accuracy on a train set: 0.840197
- For stacking approach: average accuracy on a test set: 0.703947

Results for predicting the Decision

Stacking approach and h2o AutoML results

- For stacking approach: average accuracy on a train set: 0.840197
- For stacking approach: average accuracy on a test set: 0.703947
- For h2o AutoML: average accuracy on a train set: 0.793297

Results for predicting the Decision

Stacking approach and h2o AutoML results

- For stacking approach: average accuracy on a train set: 0.840197
- For stacking approach: average accuracy on a test set: 0.703947
- For h2o AutoML: average accuracy on a train set: 0.793297
- For h2o AutoML: average accuracy on a test set: 0.703947

Conclusion about predicting the decision variable

Model	Accuracy
Stochastic Gradient Decent	0.743421
Adaptive Boosting Classifier	0.710526
h2o AutoML	0.703947
Decision Tree	0.703947
Stacking(Mixed) Model	0.703947
Random Forest	0.697368
XG Boost Classifier	0.697368
Light GBM	0.684211
Linear Support Vector Machines	0.644737
Support Vector Machines	0.625000
Logistic Regression	0.592105
KNN	0.513158

Estimating the RATING variable

Model	Score
LASSO Regression	0.793411
Epsilon-Support Vector Regression	0.809163
Huber Regressor	0.811865
Ridge Regression	0.813751
SGD	0.817268
Elastic Net	0.825020
Linear Regression	0.831060
Thielsen Regressor	0.833203
Random Forest regressor	0.862422
Light GBM	0.872895
XGBoost Regressor	0.878517
Gradient Boosting	0.924299
Kernel Ridge Regression	1.056997
Decision Tree Regressor	1.164290

Figure: These results are on a train set

Results for predicting the Rating

Stacking approach and h2o AutoML results

- For stacking approach: RMSE score on a train data: 0.791428

Results for predicting the Rating

Stacking approach and h2o AutoML results

- For stacking approach: RMSE score on a train data: 0.791428
- For stacking approach: RMSE score on a test data: 0.811798

Results for predicting the Rating

Stacking approach and h2o AutoML results

- For stacking approach: RMSE score on a train data: 0.791428
- For stacking approach: RMSE score on a test data: 0.811798
- For h2o AutoML: RMSE score on a train data: 0.770774

Results for predicting the Rating

Stacking approach and h2o AutoML results

- For stacking approach: RMSE score on a train data: 0.791428
- For stacking approach: RMSE score on a test data: 0.811798
- For h2o AutoML: RMSE score on a train data: 0.770774
- For h2o AutoML: RMSE score on a test data: 0.840311

Results for predicting the Rating

Using other h2o models

- For GBM: RMSE score on a train data: 0.679267

Results for predicting the Rating

Using other h2o models

- For GBM: RMSE score on a train data: 0.679267
- For GBM: RMSE score on a test data: 0.838749

Results for predicting the Rating

Using other h2o models

- For GBM: RMSE score on a train data: 0.679267
- For GBM: RMSE score on a test data: 0.838749
- For RF: RMSE score on a train data: 0.735665

Results for predicting the Rating

Using other h2o models

- For GBM: RMSE score on a train data: 0.679267
- For GBM: RMSE score on a test data: 0.838749
- For RF: RMSE score on a train data: 0.735665
- For RF: RMSE score on a test data: 0.838040

Results for predicting the Rating

Using other h2o models

- For GBM: RMSE score on a train data: 0.679267
- For GBM: RMSE score on a test data: 0.838749
- For RF: RMSE score on a train data: 0.735665
- For RF: RMSE score on a test data: 0.838040
- For deep learning with 3 hidden layers and 128 neurons: RMSE score on a train data: 0.573881

Results for predicting the Rating

Using other h2o models

- For GBM: RMSE score on a train data: 0.679267
- For GBM: RMSE score on a test data: 0.838749
- For RF: RMSE score on a train data: 0.735665
- For RF: RMSE score on a test data: 0.838040
- For deep learning with 3 hidden layers and 128 neurons: RMSE score on a train data: 0.573881
- For deep learning with 3 hidden layers and 128 neurons: RMSE score on a test data: 0.895573

Conclusion about predicting the Rating variable

Name of the model	RMSE
Stacking Approach(Blending)	0.817234
H2O GBM	0.838101
H2O RF	0.838119
H2O AutoML	0.840311
H2O DL	0.895573

Figure: RMSE scores

Conclusion

- The results of this analysis point to a common choice of the most relevant predictor variables:
  - Applicant's age
  - Applicant's GPA
  - Applicant's choice of emphasis area for study in graduate school
  - Applicant's GRE scores
  - Applicant's undergraduate major
  - Applicant's average ratings of knowledge given by applicant's recommenders
- In both cases, predicting the decision and predicting the rating variable, we see that some simpler models gave us better result comparing to more complicated models. One possible explanation of this could be the relationship between our variables and the response is linear. Another explanation would be the number of sample is not too many.

Further Comments

- For the future work, one might try multiple imputation technique instead of basic method to fill the missing values.

Further Comments

- For the future work, one might try multiple imputation technique instead of basic method to fill the missing values.
- There can be extra variables that represent the number of publications, and where they are published and impact factor of journals.

Further Comments

- For the future work, one might try multiple imputation technique instead of basic method to fill the missing values.
- There can be extra variables that represent the number of publications, and where they are published and impact factor of journals.
- Having distributions of the GPA's of the universities where students are coming from may lead to have better prediction of the decision about students.

Further Comments

- For the future work, one might try multiple imputation technique instead of basic method to fill the missing values.
- There can be extra variables that represent the number of publications, and where they are published and impact factor of journals.
- Having distributions of the GPA's of the universities where students are coming from may lead to have better prediction of the decision about students.
- It would also be useful to have a quantative opinion of the strength of each recommender.

Thank you!