Lavanya.ai

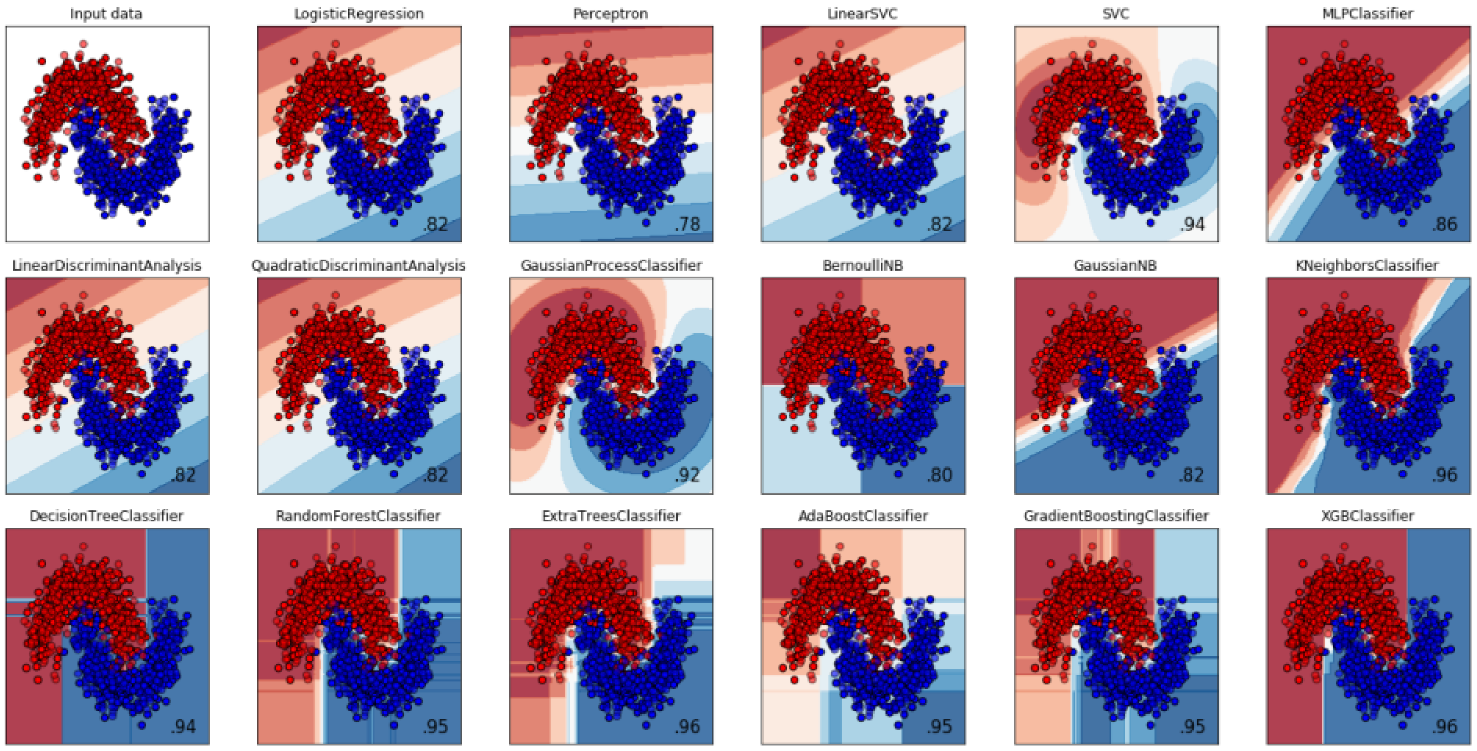
Menu

MORE INFO

ALL DEEP DIVES

Part II – A Whirlwind Tour of Machine Learning Models

BY LAVANYA
SEPTEMBER 18, 2019
COMMENTS 0



In Part I, **Best Practices for Picking a Machine Learning Model (https://lavanya.ai/2019/09/18/part-i-best-practices-for-picking-a-machine-learning-model/)**, we talked about the part art, part science of picking the perfect machine learning model.

In Part II, we dive deeper into the different machine learning models you can train and when you should use them!

In general tree-based models perform best in Kaggle competitions. The other models make great candidates for ensembling. For computer vision challenges, CNNs outperform everything. For natural language processing, LSTMs or GRUs are your best bet! With that said, below is a non-exhaustive laundry list of models to try, along with some context for each model.

1. Regression – Predicting continuous values

A. Linear Regression

- **I. Vanilla Linear Regression**

- **Advantages**
  - Captures linear relationships in the dataset well
  - Works well if you have a few well defined variables and need a simple predictive model
  - Fast training speed and prediction speeds
  - Does well on small datasets
  - Interpretable results, easy to explain
  - Easy to update the model when new data comes in
  - No parameter tuning required (the regularized linear models below need to tune the regularization parameter)
  - Doesn't need feature scaling (the regularized linear models below need feature scaling)
  - If dataset has redundant features, linear regression can be unstable
- **Disadvantages**
  - Doesn't work well for non-linear data
  - Low(er) prediction accuracy
  - Can overfit (see regularized models below to counteract this)
  - Doesn't separate signal from noise well – cull irrelevant features before use
  - Doesn't learn feature interactions in the dataset

- **II. Lasso, Ridge, Elastic-Net Regression**

- **Advantages**
  - These models are linear regression with regularization
  - Help counteract overfitting
  - These models are much better at generalizing because they are simpler
  - They work well when we only care about a few features
- **Disadvantages**
  - Need feature scaling
  - Need to tune the regularization parameter

B. Regression Trees

- **I. Decision Tree**

- **Advantages**
  - Fast training speed and prediction speeds

- Captures non-linear relationships in the dataset well
- Learns feature interactions in the dataset
- Great when your dataset has outliers
- Great for finding the most important features in the dataset
- Doesn’t need feature scaling
- Decently interpretable results, easy to explain

- **Disadvantages**
  - Low(er) prediction accuracy
  - Requires some parameter tuning
  - Doesn’t do well on small datasets
  - Doesn’t separate signal from noise well
  - Not easy to update the model when new data comes in
  - Used very rarely in practice, use ensembled trees instead
  - Can overfit (see ensembled models below)

- **II. Ensembles(RandomForest, XGBoost, CatBoost, LightGBM)**

- **Advantages**
  - Collates predictions from multiple trees
  - High prediction accuracy – does really well in practice
  - Preferred algorithm in Kaggle competitions
  - Great when your dataset has outliers
  - Captures non-linear relationships in the dataset well
  - Great for finding the most important features in the dataset
  - Separates signal vs noise
  - Doesn’t need feature scaling
  - Perform really well on high-dimensional data
- **Disadvantages**
  - Slower training speed
  - Fast prediction speed
  - Not easy to interpret or explain
  - Not easy to update the model when new data comes in
  - Requires some parameter tuning – Harder to tune
  - Doesn’t do well on small datasets

C. Deep Learning

- **Advantages**
  - High prediction accuracy – does really well in practice
  - Captures very complex underlying patterns in the data
  - Does really well with both big datasets and those with high-dimensional data
  - Easy to update the model when new data comes in
  - The network’s hidden layers reduce the need for feature engineering remarkably
  - Is state of the art for computer vision, machine translation, sentiment analysis and speech recognition tasks
- **Disadvantages**
  - Very long training speed
  - Need a huge amount of computing power
  - Need feature scaling

  - Not easy to explain or interpret results
  - Need lots of training data because it learns a vast number of parameters
  - Outperformed by Boosting algorithms for non-image, non-text, non-speech tasks
  - Very flexible, come with lots of different architecture building blocks, thus require expertise to design the architecture

D. K Nearest Neighbors – Distance Based

- **Advantages**
  - Fast training speed
  - Doesn't need much parameter tuning
  - Interpretable results, easy to explain
  - Works well for small datasets (<100k training set)
- **Disadvantages**
  - Low(er) prediction accuracy
  - Doesn't do well on small datasets
  - Need to pick a suitable distance function
  - Needs feature scaling to work well
  - Prediction speed grows with size of dataset
  - Doesn't separate signal from noise well – cull irrelevant features before use
  - Is memory intensive because it saves every observation
  - Also means they don't work well with high-dimensional data

2. Classification – Predict a class or class probabilities

A. Logistic Regression

- **Advantages**
  - Classifies linearly separable data well
  - Fast training speed and prediction speeds
  - Does well on small datasets
  - Decently interpretable results, easy to explain
  - Easy to update the model when new data comes in
  - Can avoid overfitting when regularized
  - Can do both 2 class and multiclass classification
  - No parameter tuning required (except when regularized, we need to tune the regularization parameter)
  - Doesn't need feature scaling (except when regularized)
  - If dataset has redundant features, linear regression can be unstable
- **Disadvantages**
  - Doesn't work well for non-linearly separable data
  - Low(er) prediction accuracy
  - Can overfit (see regularized models below)
  - Doesn't separate signal from noise well – cull irrelevant features before use
  - Doesn't learn feature interactions in the dataset

B. Support Vector Machines – Distance based

- **Advantages**
  - High prediction accuracy
  - Doesn't overfit, even on high-dimensional datasets, so its great for when you have lots of features
  - Works well for small datasets (<100k training set)
  - Work well for text classification problems
- **Disadvantages**
  - Not easy to update the model when new data comes in
  - Is very memory intensive
  - Doesn't work well on large datasets
  - Not easy to update the model when new data comes in
  - Requires you choose the right kernel in order to work
    - The linear kernel models linear data and works fast
    - The non-linear kernels can model non-linear boundaries and can be slow
  - Use Boosting instead!

C. Naive Bayes – Probability based

- **Advantages**
  - Performs really well on text classification problems
  - Fast training speed and prediction speeds
  - Does well on small datasets
  - Separates signal from noise well
  - Performs well in practice
  - Simple, easy to implement
  - Works well for small datasets (<100k training set)
  - The naive assumption about the independence of features and their potential distribution lets it avoid overfitting
  - Also if this condition of independence holds, Naive Bayes can work on smaller datasets and can have faster training speed
  - Doesn't need feature scaling
  - Not memory intensive
  - Decently interpretable results, easy to explain
  - Scales well with the size of the dataset
- **Disadvantages**
  - Low(er) prediction accuracy

D. K Nearest Neighbors – Distance Based

- **Advantages**
  - Fast training speed
  - Doesn't need much parameter tuning
  - Interpretable results, easy to explain
  - Works well for small datasets (<100k training set)
- **Disadvantages**
  - Low(er) prediction accuracy
  - Doesn't do well on small datasets
  - Need to pick a suitable distance function
  - Needs feature scaling to work well

  - Prediction speed grows with size of dataset
  - Doesn't separate signal from noise well – cull irrelevant features before use
  - Is memory intensive because it saves every observation
  - Also means they don't work well with high-dimensional data

E. Classification Tree

- I. **Decision Tree**

- **Advantages**
  - Fast training speed and prediction speeds
  - Captures non-linear relationships in the dataset well
  - Learns feature interactions in the dataset
  - Great when your dataset has outliers
  - Great for finding the most important features in the dataset
  - Can do both 2 class and multiclass classification
  - Doesn't need feature scaling
  - Decently interpretable results, easy to explain
- **Disadvantages**
  - Low(er) prediction accuracy
  - Requires some parameter tuning
  - Doesn't do well on small datasets
  - Doesn't separate signal from noise well
  - Used very rarely in practice, use ensembled trees instead
  - Not easy to update the model when new data comes in
- Can overfit (see ensembled models below)

- II. **Ensembles(RandomForest, XGBoost, CatBoost, LightGBM)**

- **Advantages**
  - Collates predictions from multiple trees
  - High prediction accuracy – does really well in practice
  - Preferred algorithm in Kaggle competitions
  - Captures non-linear relationships in the dataset well
  - Great when your dataset has outliers
  - Great for finding the most important features in the dataset
  - Separates signal vs noise
  - Doesn't need feature scaling
  - Perform really well on high-dimensional data
- **Disadvantages**
  - Slower training speed
  - Fast prediction speed
  - Not easy to interpret or explain
  - Not easy to update the model when new data comes in
  - Requires some parameter tuning – Harder to tune
  - Doesn't do well on small datasets

F. Deep Learning

- **Advantages**

  - High prediction accuracy – does really well in practice
  - Captures very complex underlying patterns in the data
  - Does really well with both big datasets and those with high-dimensional data
  - Easy to update the model when new data comes in
  - The network's hidden layers reduce the need for feature engineering remarkably
  - Is state of the art for computer vision, machine translation, sentiment analysis and speech recognition tasks
- **Disadvantages**
  - Very long training speed
  - Not easy to explain or interpret results
  - Need a huge amount of computing power
  - Need feature scaling
  - Need lots of training data because it learns a vast number of parameters
  - Outperformed by Boosting algorithms for non-image, non-text, non-speech tasks
  - Very flexible, come with lots of different architecture building blocks, thus require expertise to design the architecture

3. Clustering – Organize the data into groups to maximize similarity

A. DBSCAN

- **Advantages**
  - Scalable to large datasets
  - Detects noise well
  - Don't need to know the number of clusters in advance
  - Doesn't make an assumption that the shape of the cluster is globular
- **Disadvantages**
  - Doesn't always work if your entire dataset is densely packed
  - Need to tune the density parameters – epsilon and min_samples to the right values to get good results

B. KMeans

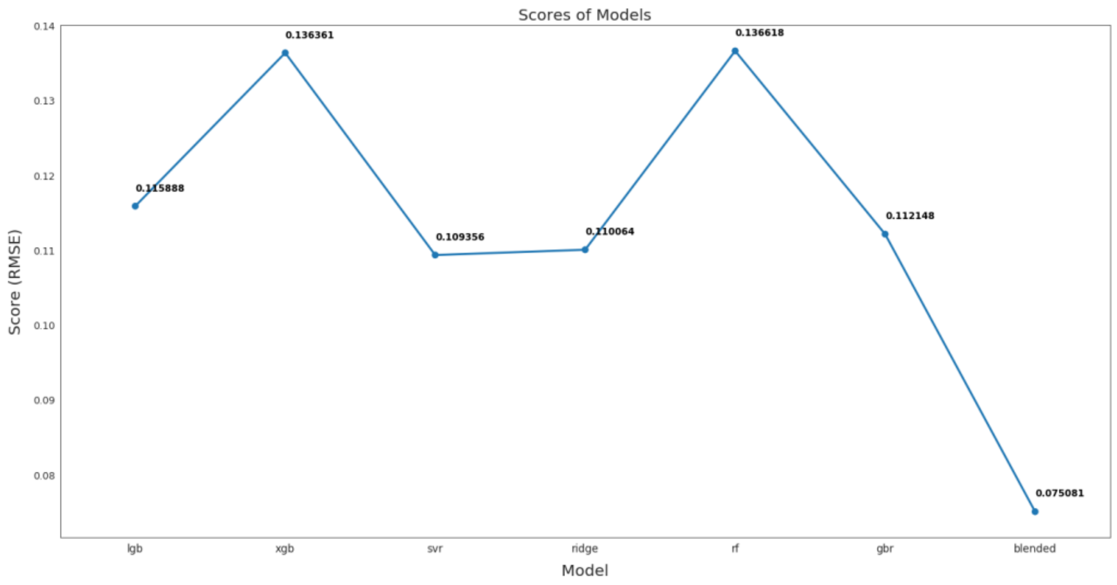
- **Advantages**
  - Great for revealing the structure of the underlying dataset
  - Simple, easy to interpret
  - Works well if you know the number of clusters in advance
- **Disadvantages**
  - Doesn't always work if your clusters aren't globular and similar in size
  - Needs to know the number of clusters in advance – Need to tune the choice of k clusters to get good results
  - Memory intensive
  - Doesn't scale to large datasets

4. Misc – Models not included in this post

- Dimensionality Reduction Algorithms
- Clustering algorithms
  - Gaussian Mixture Model
  - Hierarchical clustering
- Computer Vision (CV)
  - Convolutional Neural Networks
  - Image classification
  - Object Detection
  - Image segmentation
- Natural Language Processing (NLP)
  - RNNs (LSTM or GRUs)
- Reinforcement Learning

Ensembling Your Models

Ensembling models is a really powerful technique that helps reduce overfitting, and make more robust predictions by combining outputs from different models. It is especially an essential tool for winning Kaggle competitions.When picking models to ensemble together, we want to pick them from different model classes to ensure they have different strengths and weaknesses and thus capture different patterns in the dataset. This greater diversity leads to lower bias. We also want to make sure their performance is comparable in order to ensure stability of predictions generated.We can see here that the blending these models actually resulted in much lower loss than any single model was able to produce alone. Part of the reason is that while all these models are pretty good at making predictions, they get different predictions right and by combining them together, we're able to combine all their different strengths into a super strong model.



```python
# Blend models in order to make the final predictions more robust to ove
def blended_predictions(X):
    return ((0.1 * ridge_model_full_data.predict(X)) + \\
            (0.2 * svr_model_full_data.predict(X)) + \\
            (0.1 * gbr_model_full_data.predict(X)) + \\
            (0.1 * xgb_model_full_data.predict(X)) + \\
            (0.1 * lgb_model_full_data.predict(X)) + \\
            (0.05 * rf_model_full_data.predict(X)) + \\
            (0.35 * stack_gen_model.predict(np.array(X))))
```

There are 4 types of ensembling (including blending):

- **Bagging:** Train many base models with different randomly chosen subsets of data, with replacement. Let the base models vote on final predictions. Used in RandomForests.
- **Boosting:** Iteratively train models and update the importance of getting each training example right after each iteration. Used in GradientBoosting.
- **Blending:** Train many different types of base models and make predictions on a holdout set. Train a new model out of their predictions, to make predictions on the test set. (Stacking with a holdout set).
- **Stacking:** Train many different types of base models and make predictions on k-folds of the dataset. Train a new model out of their predictions, to make predictions on the test set.

Comparing Models

Weights and Biases lets you track and compare the performance of you models with one line of code.Once you have selected the models you'd like to try, train them and simply add *wandb.log({'score': cv_score})* to log your model state. Once you're done training, you can compare your model performances in one easy dashboard!

```python
# WandB
import wandb
import tensorflow.keras
```

```
from wandb.keras import WandbCallback
from sklearn.model_selection import cross_val_score
# Import models (Step 1: add your models here)
from sklearn import svm
from sklearn.linear_model import Ridge, RidgeCV
from xgboost import XGBRegressor

# Model 1
# Initialize wandb run
# You can change your project name here. For more config options, see h
wandb.init(anonymous='allow', project="pick-a-model")

# Initialize model (Step 2: add your classifier here)
clf = svm.SVR(C= 20, epsilon= 0.008, gamma=0.0003)

# Get CV scores
cv_scores = cross_val_score(clf, X_train, train_labels, cv=5)

# Log scores
for cv_score in cv_scores:
    wandb.log({'score': cv_score})

# Model 2
# Initialize wandb run
# You can change your project name here. For more config options, see h
wandb.init(anonymous='allow', project="pick-a-model")

# Initialize model (Step 2: add your classifier here)
clf = XGBRegressor(learning_rate=0.01,
                       n_estimators=6000,
                       max_depth=4,
                       min_child_weight=0,
                       gamma=0.6,
                       subsample=0.7,
                       colsample_bytree=0.7,
                       objective='reg:linear',
                       nthread=-1,
                       scale_pos_weight=1,
                       seed=27,
                       reg_alpha=0.00006,
                       random_state=42)

# Get CV scores
cv_scores = cross_val_score(clf, X_train, train_labels, cv=5)

# Log scores
for cv_score in cv_scores:
    wandb.log({'score': cv_score})

# Model 3
# Initialize wandb run
# You can change your project name here. For more config options, see h
wandb.init(anonymous='allow', project="pick-a-model")

# Initialize model (Step 2: add your classifier here)
ridge_alphas = [1e-15, 1e-10, 1e-8, 9e-4, 7e-4, 5e-4, 3e-4, 1e-4, 1e-3,
clf = Ridge(alphas=ridge_alphas)
```

```python

# Get CV scores
cv_scores = cross_val_score(clf, X_train, train_labels, cv=5)

# Log scores
for cv_score in cv_scores:
    wandb.log({'score': cv_score})
```

That's it now you have all the tools you need to pick the right models for your problem!

Model selection and can be very complicated, but I hope this guide sheds some light and gives you a good framework for picking models.

If you have any questions, please don't hesitate to **tweet me (https://twitter.com/lavanyaai)**.

0 comments on "Part II – A Whirlwind Tour of Machine Learning Models"

Lavanya.ai