Kevin Coffey
Lecture Notes

Applied Machine Learning

Supervised machine learning (Part 2)

Kevin Coffey Thompson
Associate Professor of Information & Computer Science
University of Michigan

Applied Machine Learning

Naive Bayes Classifiers

- These classifiers are called Naive Bayes because they assume that features are conditionally dependent given the class. In other words, they assume that the probability of a feature, given the class, is the Bayesian Naive Bayes classifiers can also allow for continuous features, but they must assume that the probability of continuous features is Gaussian.
- Despite the strong assumption, Naive Bayes classifiers work well in practice.
- Can be competitive for some tasks.

Naive Bayes classifier types

- **Bernoulli**: binary features (e.g. word occurrences in documents)
- **Multinomial**: discrete features (e.g. word counts)
- Gaussian: continuous/real valued features
  - Assumes each feature is distributed according to a Gaussian distribution

- **See the Applied Text Mining course for more details on the Bernoulli and Multinomial Naive Bayes classifiers.**

Gaussian Naive Bayes classifier

Gaussian Naive Bayes classifier

Gaussian Naive Bayes classifier

Naive Bayes classifiers: Pros and Cons

Pros	Cons
Simple, efficient	Assumption of conditional independence
Can handle continuous features and other data types	Gaussian assumption may be inappropriate
Often useful as a baseline	Zero probability problem
Competitive for many tasks	Poor probability estimates
Particularly useful for text classification	Sensitive to irrelevant features
	Sensitive to outliers

Applied Machine Learning

Random Forests

Random Forest Process

Random Forest Process: Bootstrap Samples

Random Forest Process

Random Forest `max_features` Parameter

- Learning is quite sensitive to `max_features`.
- Setting `max_features` to `None` is fitting with depth, more complex trees.
- Setting `max_features` to `1` is fitting with `max_features` will lead to similar forests with simpler trees.

Prediction Using Random Forests

1. Make a prediction for every tree in the forest.
2. Combine individual predictions:
   - Regression: mean of individual trees
   - Classification:
     - Hard voting:
       - Each tree votes probability for each class.
       - Pick the class with the most votes.
     - Soft voting:
       - Average the probability for each class.
       - Pick the class with highest probability.

Random Forest: Fruit Dataset

Random Forest: Pros and Cons

Pros	Cons
Robust, not sensitive to irrelevant features, outliers, or correlated features	Less interpretable than single trees
Handles mixed data types	Slower than single trees, memory intensive for large forests
Provides feature importance scores	Hyperparameter tuning can be time-consuming
Ensemble reduces overfitting compared to single trees	Bootstrap sampling reduces training data per tree

Random Forests `RandomForestClassifier` Hyperparameters

- `n_estimators`: number of trees to use in ensemble (default: 10).
- `max_features`: number of features to consider when looking for the best split (default: sqrt(n_features)).
- `max_depth`: maximum depth of the tree (default: None).
- `min_samples_split`: minimum number of samples required to split an internal node (default: 2).
- `bootstrap`: whether bootstrap samples are used when building trees (default: True).
- `n_jobs`: number of jobs to run in parallel (default: None).
- `random_state`: controls the random seed for reproducibility.
- Choose a large value for the `n_estimators` parameter if you need more accurate predictions.

Applied Machine Learning

Gradient Boosted Decision Trees

Kevin Coffey Thompson
Associate Professor of Information & Computer Science
University of Michigan

Gradient Boosted Decision Trees (GBDT)

GBDT: Pros and Cons

Pros	Cons
Very high accuracy on structured/tabular data	Sensitive to outliers (squared loss amplifies their influence)
Handles mixed data types (numerical, categorical)	Requires careful hyperparameter tuning
Provides feature importance scores	Sequential training limits parallelization
Can handle missing values (XGB, LGB, CatBoost)	Memory intensive for large datasets

GBDT: `GradientBoostingClassifier` Hyperparameters

- `n_estimators`: number of weak learners to use (base learners) in the ensemble.
- `learning_rate`: controls how much each tree contributes to the ensemble.
- `max_depth`: maximum depth of each weak learner.
- `subsample`: fraction of samples used for each weak learner (default: 1.0 = all samples).
- `random_state`: controls the random seed for reproducibility.
- Smaller `learning_rate` values need more `n_estimators` to achieve the same training error.

Applied Machine Learning

Neural networks

Kevin Coffey Thompson
Associate Professor of Information & Computer Science
University of Michigan

Review: Linear and Logistic Regression

MLP/Sac Perceptron with two hidden layer activation function

MLP/Sac Perceptron with two hidden layer activation function

Activation Functions

A single hidden layer network using 1, 10, or 100 units

Multi-layer Perceptron with Two Hidden Layers

One vs Two Hidden Layers

L2 Regularization with the Alpha Parameter

Neural Network Regression with MLPRegressor

Neural Networks: Pros and Cons

Pros	Cons
Can learn the boundary of complex or non-linear data	Larger models require more training data to avoid overfitting
Can be more efficient than advanced tree/ensemble methods on very large datasets	Training and tuning can be time-consuming
Can be more accurate than other methods	Black-box nature makes it difficult to explain predictions compared to simpler models

Neural Nets: `MLPClassifier` and `MLPRegressor` Hyperparameters

- `hidden_layer_sizes`: sets the number of hidden layers and the number of units in each layer. Longer lists are faster than shorter lists.
- `alpha`: controls weight on the regularization penalty (default: 0.0001). Higher values lead to smaller weights.
- `activation`: controls the nonlinear function used for hidden layers (`relu`, `tanh`, `logistic`, `identity`).
- `random_state`: controls the random seed for reproducibility.
- All inputs are used in the first supervised learning course.

Applied Machine Learning

Deep Learning

Kevin Coffey Thompson
Associate Professor of Information & Computer Science
University of Michigan

Deep Learning Summary

- Deep learning architectures combine a predifined/learned feature extraction pipeline with a supervised learning model.
- Deep learning models are composed of multiple hidden layers which learn representations of increasing complexity.
- Deep learning models are data hungry and, in general, learn their feature hierarchies from data.
- Deep learning models are typically trained using SGD.
- Activations are used in the hidden layers.

An example of a simple deep learning model

Deep Learning

Deep Learning

Pros and Cons of Deep Learning

Pros	Cons
Deep learning models can learn feature representations from data.	Deep learning models are data hungry and can underperform on small datasets.
Deep learning models can be trained using SGD and can be trained on very large datasets.	Training deep learning models can be time-consuming and requires specialized hardware (GPUs).
Deep learning models can be adapted to many different tasks.	Deep learning models are difficult to tune and have many hyperparameters.
Deep learning models can be pre-trained and fine-tuned for new tasks.	Deep learning models are black-box models and difficult to interpret.

Deep learning software for Python

- **Keras**: https://keras.io/
- **Lasagne**: https://lasagne.readthedocs.io/en/latest/
- **TensorFlow**: https://www.tensorflow.org/
- **PyTorch**: https://pytorch.org/docs/stable/index.html
- Libraries support high-performance computation via GPUs.

Applied Machine Learning

Data Leakage

Kevin Coffey Thompson
Associate Professor of Information & Computer Science
University of Michigan

Data Leakage

- **When the data you're using to make predictions contains information about what you're trying to predict, but this information wouldn't be available at prediction time (or would not be available in the future).**
- **Data leakage can lead to inflated performance estimates and poor generalization.**
- Data leakage can be subtle and difficult to detect.
- Including the target in the feature set is a clear example of data leakage.
- **If you are suspicious of a feature, it is good to be safe. It is better to be safe than sorry.**
- **Data leakage can be difficult to detect if you don't understand the data and how it was collected.**
- **Data leakage can be difficult to detect if you don't understand the problem you are trying to solve.**

More subtle examples of data leakage

- Including information about the future when predicting the present (e.g., using tomorrow's stock price to predict today's price).
- Including information from the test set in the training set (e.g., fitting a scaler on the entire dataset before splitting into train and test).
- Including information from the test set in the training set (e.g., using cross-validation to tune hyperparameters but then reporting the cross-validation score as the final test set score).

Other examples of data leakage

- Including information about the future when predicting the present (e.g., using tomorrow's stock price to predict today's price).
- Including information from the test set in the training set (e.g., fitting a scaler on the entire dataset before splitting into train and test).
- Including information from the test set in the training set (e.g., using cross-validation to tune hyperparameters but then reporting the cross-validation score as the final test set score).

Detecting data leakage

- Before building the model:
  - Understand the data and how it was collected.
  - Understand the problem you are trying to solve.
  - Be suspicious of features that seem too good to be true.
  - Be suspicious of features that are highly correlated with the target.
  - Be suspicious of features that are highly correlated with other features.
- After building the model:
  - Be suspicious of high accuracy scores.
  - Be suspicious of high feature importance scores.
  - Be suspicious if the model performs well on the training data but poorly on the test data.
  - Be suspicious if the model performs well on the test data but poorly on new data.

Applied Machine Learning

Minimizing Data Leakage

- Perform data preprocessing within each cross-validation fold separately.
- Use `Pipeline` to ensure that preprocessing is applied correctly within each cross-validation fold.
- Be careful with time-series data, ensure that you are not using future information to predict the past.
- Be careful with `GroupKFold` or `StratifiedGroupKFold` if your data has groups (e.g., multiple samples from the same patient).
- Be careful with `TimeSeriesSplit` if your data is time-series data.
- Be careful with `StratifiedKFold` if the class ratio changes between training and test sets.