Assignment 1 Analysis  
ML 7641

Ryan Wingate

RWingate8

GTID# 903453413

Implementation Strategy

For this assignment, I used the scikit-learn Python library, specifically, version 0.21.2. Additional dependencies include Jupyter Notebook (4.5.0) for convenient iteration and inline documentation of code, pandas (0.25.1) for data manipulation and cleaning, and Matplotlib (3.1.1) for charts and visualization.

The package manager I used was Anaconda (4.7.11), and the Python version was 3.7.4. For additional details on replicating the environment used for this assignment, reference the README included with this report.

Specific Algorithm Implementations

For each of the algorithms outlined below, I used the appropriate scikit-learn implementation.

Decision Tree

DecisionTreeClassifier, imported via:

from sklearn import tree

Neural Network

MLPClassifier, imported via:

from sklearn.neural\_network import MLPClassifier

Boosting

AdaBoostClassifier, imported via

from sklearn.ensemble import AdaBoostClassifier

Support Vector Machine

NuSVC, imported via

from sklearn.svm import NuSVC

*K*-Nearest Neighbors

KNeighborsClassifier, imported via

from sklearn.neighbors import KNeighborsClassifier

Classification Problem Description

The problems I selected for this assignment are two famous datasets, both of which are designed for binary classification tasks. They have a similar number of features but differ in the number of available samples by a factor of roughly 50. This was intentional, so that I could experience the differences in training time between the various algorithms when the sample count varies widely.

Exact dataset preprocessing steps are available in Jupyter notebooks I provided in the submitted GitHub repository. A high-level overview of the data-preprocessing steps I completed:

• Discard irrelevant features,

• Fill nan values,

• Replace nominal and ordinal categorical data with numeric representations,

• Create dummy variables for all nominal categorical data with 3 or more possible categories (dropping one column, to prevent multicollinearity), and

• Normalize continuous data.

Following processing, for both datasets, I randomly extracted 20% of the data using scikit-learn’s train\_test\_split function. I used the remaining 80% as training data. I used 20% of the training data as a cross-validation set.

Titanic Dataset

The first dataset is the well-known Titanic Survival dataset, downloaded from <https://www.kaggle.com/c/titanic>. I consider the dataset interesting for a few reasons:

• Ability to successfully learn a variety of ML algorithms, despite its small size,

• Accuracy values empirically shown to be in the high 70s to mid 80s, which falls into the “interesting” range for most datasets,

• Subject matter provides interesting social commentary,

• Possibility of deriving interesting features from raw data (Deck from Cabin, for example),

• Widely considered by the data science community as being “interesting,”

The raw Titanic dataset consists of 1309 samples with 12 features.

Table 1: Raw Titanic dataset features and types.

|  |  |
| --- | --- |
| **Feature** | **Data Type** |
| PassengerId | Integer |
| Pclass | Category (3 value) |
| Name | String |
| Sex | Category (2 value) |
| Age | Float |
| SibSp | Integer |
| Par/Ch | Integer |
| Ticket | String |
| Fare | Float |
| Cabin | String |
| Embarked | Category (3 value) |
| Survived | Category (2 value) |

I discarded the **Ticket**, **Name**, and **PassengerID** features as non-useful data, and used **Cabin** (“C123”) to derive an 8-value classification **Deck** (“C”), where possible, then I discarded **Cabin** as well. **Sibs** and **Par/Ch** were the count of siblings and count of parents/children also on the boat, respectively. **Sibs**, **Par/Ch**, and **Pclass** were interpreted as Ordinal categories and left as integers in the data. I interpreted **Survived**, **Pclass**, **Sex**, **Embarked**, and **Deck** were interpreted as nominal categories and processed them into dummy variables. I normalized **Age** and **Fare**.

Table 2: Processed Titanic dataset features and types.

|  |  |
| --- | --- |
| **Feature** | **Processed Data Type** |
| Pclass | Ordinal Category |
| Sex | Nominal Category (2) |
| Age | Float |
| SibSp | Ordinal Category |
| Parch | Ordinal Category |
| Fare | Float |
| Deck | Nominal Category (8) |
| Embarked | Nominal Category (3) |
| Survived | Nominal Category (2) |

Following processing, the dataset consisted of 1309 rows and 17 columns, which describe 8 input features and 1 output feature.

The task for the Titanic dataset is to predict whether or not a given passenger will survive the Titanic disaster. This is indicated by an appropriate value of the **survived** feature.

Adult Dataset

The second dataset is the Adult dataset, downloaded from <https://archive.ics.uci.edu/ml/datasets/Adult>. I consider the dataset interesting for a few reasons:

• Relatively large size is an interesting point of comparison with the small Titanic dataset,

• Accuracy values empirically shown to be in the high 70s to low 80s, which falls into the “interesting” range for most datasets,

• Personal interest in the factors driving compensation,

• Availability of another, larger but very similar dataset (“Census,” 200K+ samples) with similar information and actual compensation values, should I need more, or more precise, data, and

• Widely considered by the data science community as being “interesting,” as evidenced by 1.6 million web hits since 2007, making it the second most popular dataset on the UCI ML dataset repository.

The raw Adult dataset consists of 46012 samples and 14 features.

Table 3: Raw Adult dataset features and types.

|  |  |
| --- | --- |
| **Feature** | **Data Type** |
| age | Integer |
| employment-type | Category (8) |
| fnlwgt | Float |
| education | Category (15) |
| education-num | Integer |
| marital-status | Category (7) |
| occupation | Category (14) |
| relationship | Category (6) |
| race | Category (5) |
| sex | Category (2) |
| capital-gain | Float |
| capital-loss | Float |
| weekly-hours | Float |
| native-country | Category (41) |
| compensation | Category (2) |

To reduce the overall dimensionality of the dataset, I also discard **fnlwgt**, **marital-status**, **relationship**, **capital-gain**, **capital-loss**, and **native-country**. I also discard **education-num** as being redundant with **education**. I exclude all samples that do not have a **workclass** or **occupation** assigned, or which have a **workclass** that has value “Without-pay,” as all of these would be highly correlated with low income and therefore uninteresting.

Due to the large number of available categories, I map:

• **education** from 15 categories to 6,

• **employment-type** from 8 categories to 3, and

• **occupation** from 14 categories to 13.

I interpreted **employment-type**, **occupation**, and **race** as nominal categories and processed them into dummy variables. I normalized **age** and **weekly-hours**.

Table 4: Processed Adult dataset features and types.

|  |  |
| --- | --- |
| **Feature** | **Processed Data Type** |
| age | Integer |
| employment-type | Nominal Category (3) |
| education | Ordinal Category (6) |
| occupation | Nominal Category (13) |
| race | Nominal Category (5) |
| sex | Nominal Category (2) |
| weekly-hours | Float |
| compensation | Nominal Category (2) |

Following processing, the dataset consisted of 46,012 rows and 24 columns, which describe a total of 7 input features and one output feature.

The task for the Adult dataset is to predict whether a given person earns more than $50K. This is indicated by an appropriate value of the **compensation** feature.

Hyperparameter Tuning

The rough heuristic I employ throughout the following sections is that a performance gap of less than 1% between training and cross-validation datasets constitutes an “acceptable” fit to the data. A gap of 1% to 3% is “marginal,” and may be acceptable depending on context. A gap larger than 3% is deemed “unacceptable” as it indicates an overfit.

Decision Tree

Titanic Dataset

Figure 1-1 shows the learning curve for a default decision tree trained on the Titanic dataset. The model exhibits the high variance characteristic of an overfit. This is expected as the default hyperparameters essentially allows the decision tree to grow arbitrarily complex.

A close up of text on a white background

Description automatically generated

Fig 1.1

The “entropy” splitting criterion appears to improve more monotonically with added training examples.

A close up of text on a white background

Description automatically generated

Fig 1.2

The “entropy” splitting criterion appears to improve more monotonically with added training examples. Lacking a reason to use the “gini” splitting criterion, I move forward with familiar “entropy.”

The pruning method I implement uses the max\_leaf\_nodes hyperparameter. I select it because it is a less blunt restriction than the more common max\_depth hyperparameter in that it enables the tree to increase splitting complexity and layer count where it would improve model performance.

A close up of a map

Description automatically generated

Fig 1.3

The cross-validation score maximizes at 87.3% with a max\_leaf\_nodes value of 10. The shape of the curve in the region surrounding 10 implies a finer-grained search may be useful.

A screenshot of a cell phone

Description automatically generated

Fig 1.4

Again, the cross-validation score maximizes at 87.3% and an acceptable delta of 0.7%, this time with a max\_leaf\_nodes value of 8.

A close up of a map

Description automatically generated

Fig 1.5

Adult Dataset

To preserve a common approach with the Titanic dataset training process, I begin with the ‘entropy’ splitting criterion. Figure 2.1 shows the learning curve for an otherwise default decision tree trained on the Adult dataset. The model exhibits high variance, characteristic of an overfit.

A screenshot of a cell phone

Description automatically generated

Fig 2.1

The cross-validations core maximizes at 80.5% with max\_leaf\_nodes = 100, but the curve is very flat across all the values shown below.

A screenshot of a cell phone

Description automatically generated

Fig 2.2

Despite the relatively complex tree, the delta between training and cross-validation tests is acceptable at only 0.8%.

A screenshot of a cell phone

Description automatically generated

Fig 2.3

As a final check, I run the identical learning curve analysis, except using the ‘gini’ node splitting criterion. The plots appear very similar qualitatively, but the C.V. score is slightly worse at 80.4% and the training/C.V. delta is also worse at 1.1%.

Neural Network

Titanic Dataset

The default hyperparameters for scikit-learn’s MLPClassifier are activation = “relu”, solver = “adam”, max\_iter = 200, and hidden\_layer\_sizes = (100,), which means it uses a single hidden layer with 100 neurons.

Training a neural network with these hyperparameters produces a marginal fit to the data, with a 2.2% gap between the training and cross-validation scores.

A close up of text on a white background

Description automatically generated

Fig 3.1

Using the more familiar “logistic” activation function and “sgd” solver in combination with the default 200 max iterations results in a classifer with very high bias. That model scores roughly 65% on both training and cross-validation data.

In order to use the “logistic” activation function and “sgd” solver, it is necessary to use a larger max iteration value of 2000. These hyperparameters result in an excellent fit to the data, with a cross-validation score of 86.5% and a training score of 86.1%.

A close up of a map

Description automatically generated

Fig 3.2

Iteration demonstrates that the most accurate ombination of solver and activation function appears to be “adam” and “logistic,” respectively. It appears that the 100-unit hidden layer is overkill, despite the excellent fit to the data in Fig 3.2.

The best-performing hidden layer configuration is a single hidden layer of 12 neurons.

A screenshot of a cell phone

Description automatically generated

Fig 3.3

With this combination of parameters, the fit to the data is excellent, with a training score of 86.6%, cross-validation score of 85.9%, and a low delta of only 0.7%.

A screenshot of a cell phone

Description automatically generated

Fig 3.4

Adult Dataset

Due to the much larger dataset size, training a default neural network on the Adult dataset results in much less variance than the same neural network on the smaller Titanic dataset. Using defaut hyperparameters, the delta between the training and cross-validations score is marginal at 1.5%.

A screenshot of a cell phone

Description automatically generated

Fig 4.1

Switching to the logistic activation function increases training time, but also results in a further decrease in overall variance.

A screenshot of a cell phone

Description automatically generated

Fig 4.2

As when training the Titanic dataset, I use the relatively time-efficient default “adam” solver rather than “sgd.”

A screenshot of a cell phone

Description automatically generated

Fig 4.3

The optimal hidden layer configuration is surprisingly stable across various sizes, but the optimal value appears to be 17. It also appears that it is not necessary to increase the number of iterations per datapoint, due to the larger dataset size, in order to get a result with acceptable bias.

So configured, the network has a training score of 80.8% and a cross-validation score of 80.5%.

A screenshot of a cell phone

Description automatically generated

Fig 4.4

Boosted Decision Tree

Titanic Dataset

Figure 5.1 shows the learning curve for the default base estimator used by the scikit-learn’s AdaBoostClassifier, a single-layer decision tree.

A close up of text on a white background

Description automatically generated

Fig 5.1

Even this very simple tree has very high cross-validation and training scores, at 86.1% and 86.0%, respectively. This is due to passenger sex being highly predictive of survival. Specifically, women were much more likely to survive than men, and the single-layer tree does in fact select gender as the root node.

Experimentation shows that anything more complex than a single-layer decision tree results in immediate overfitting with even very restricted estimator counts. Further, the performance of the a two-layer decision tree is only 0.1% improved by the additional layer.

With the default number of estimators (50), the boosted decision tree exhibits a marginal fit to the data, with a training score of 88.2% and a cross-validation score of 86.4%.

A close up of a map

Description automatically generated

Fig 5.2

The boosted decision tree exhibits almost no marginal accuracy improvement with estimator counts above 20, but with slowly increasing variance. The optimal value is 6 estimators.

A screenshot of text

Description automatically generated

Fig 5.3

With 6 estimators, the cross-validation score is 86.6% and the training score is 86.7%, an excellent fit to the Titanic data.

A close up of a map

Description automatically generated

Fig 5.4

Adult Dataset

Using the Adult dataset, it is possible to include over 250 single-layer decision tree estimators in a boosting algorithm and not overfit. Specifically, with 250 single-layer decision tree estimators, cross-validation and training scores of 80.7% and 80.8%, respectively, are possible.

Iteration demonstrates that two-layer decision trees result in a boosting algorithm that operates with a more reasonable number of estimators, and which outperforms the single-layer decision tree. On its own, a two-layer decision tree has zero overfitting on the Adult dataset.

A screenshot of a cell phone

Description automatically generated

Fig 6.1

With 55 estimators, the boosted two-layer decision tree has a cross-validation accuracy of 81.0% and a training accuracy of 81.3%.

A screenshot of a cell phone

Description automatically generated

Fig 6.2

A screenshot of a cell phone

Description automatically generated

Fig 6.3

Support Vector Machine

Titanic Dataset

Scikit-learn includes a few implementations of support vector machines. For my analysis, I used the NuSVC algorithm, which includes more than three kernels including “rbf,” “linear,” and “poly.” These are shown in figures 7.1, 7.2, and 7.3, respectively.

A screenshot of a cell phone

Description automatically generated

Fig 7.1

A screenshot of a cell phone

Description automatically generated

Fig 7.2

A close up of a map

Description automatically generated

Fig 7.3

Each of these kernels had roughly similar performance when trained over the entire training set. An additional parameter exposed by the NuSVC algorithm is the Nu parameter, which is a lower bound on the fraction of support vectors and an upper bound on the fraction of training errors.

As shown in the following figures, the performance of the various kernels depends on the how that kernel interacts with the nu parameter.

A close up of a map

Description automatically generated

Fig 7.4

A close up of a map

Description automatically generated

Fig 7.5

A close up of a map

Description automatically generated

Fig 7.6

I proceed with the polynomial kernel function as it appears most sensitive to the nu factor, and therefore most likely to result in the highest level of performance following optimization. The degree of the polynomial kernel function is also a parameter of the function, and 2 appears to be the optimal value.

A close up of a map

Description automatically generated

Fig 7.7

The optimal nu for a second-degree polynomial is 0.4, which is the final configuration for the support vector machine, and which fits the data very well with cross-validation and training scores of 87.0% and 87.2% respectively.

A close up of a map

Description automatically generated

Fig 7.8

Adult Dataset

Training a support vector machine on the full Adult dataset proved was the least time-efficient training operation encountered so far. I ultimately determined that training on the complete Adult dataset (46,012 rows) was impractical for purposes of hyperparameter estimation.

Instead, I used a 15,625-row subset and found that it could train the algorithm within a few minutes, in most cases, which was much less than half of the time to train the complete dataset. A 15,625-row subset resulted in 10,000 rows of training data, after splitting out both testing and cross-validation sets. This means that the Adult training set is more than a factor of 10 larger than the Titanic training set.

The default value of n (0.50) was “infeasible” for the Adult dataset. Experimentation showed that values of Nu less than roughly 0.40 produce high variance and/or high bias, for each of the “rbf,” “linear,” and “poly” kernels.

A close up of a map

Description automatically generated

Fig 8.1

A close up of text on a white background

Description automatically generated

Fig 8.2

Nu values of 0.45 appear to be optimal across all three kernels under consideration. The “linear” kernel is clearly the lowest-performing, with a cross-validation accuracy of 75.5%. I exclude it from further consideration.

A close up of a map

Description automatically generated

Fig 8.3

The “rbf” and “poly” kernels have cross-validation scores of 81.0% and 80.2%, respectively, at a nu of 0.4, but have slightly reduced accuracy and improved variance at a nu of 0.45.

As with the Titanic dataset, I attempt to optimize the degree of the polynomial with fixed nu and note that performance is slightly improved with degree 2, and that the optimal nu is still 0.45.

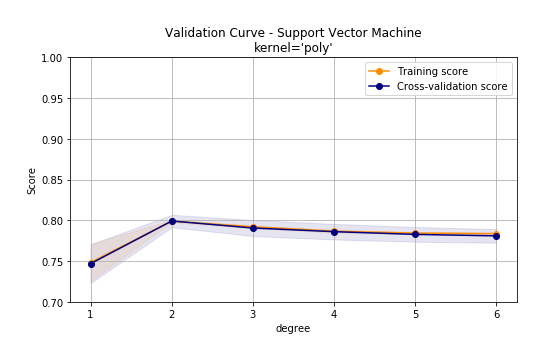


Fig 8.4

Even with these additional optimizations, the “rbf” kernel with nu = 0.45 outperformers the 2nd degree polynomial, though by a very slim margin.

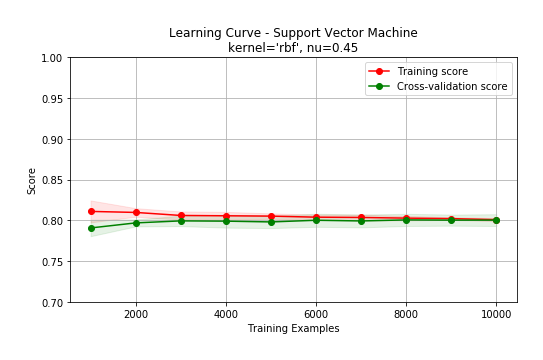


Fig 8.5

K-Nearest Neighbors

Titanic Dataset

The default hyperparameters for scikit-learn’s implementation of K-Nearest Neighbors is 5 nearest neighbors and a power parameter of 2. A power parameter of 2 is a standard Euclidean distance metric. A power parameter of 1 is Manhattan distance.

With default parameters, the K-Nearest Neighbors classifier results in an overfit, with cross-validation and training scores of 84.4 and 88.2, respectively.

A close up of a map

Description automatically generated

Fig 9.1

The optimal power parameter appears to be 1.

A screenshot of a cell phone

Description automatically generated

Fig 9.2

The optimal neighbor count appears to be 28.

A close up of a map

Description automatically generated

Fig 9.3

With a Manhattan distance metric and a nearest neighbor count of 28, the algorithm performs well, with a cross-validations core of 86.3% and a training score of 86.7%.

A close up of a map

Description automatically generated

Fig 9.4

Adult Dataset

With default hyperparameters, the K-Nearest Neighbors overfits to the training data, with a training score of 84.3% and a cross-validation score of 78.4%.

A screenshot of a cell phone

Description automatically generated

Fig 10.1

As before, Manhattan distance appears to be the optimal distance metric, though by a very slim margin.

A screenshot of a cell phone

Description automatically generated

Fig 10.2

With the distance metric specified as Manhattan distance, the optimal neighbor count appears to be 60.

A screenshot of a cell phone

Description automatically generated

Fig 10.3

The cross-validation and training scores are 80.6% and 81.2%, an excellent fit to the data.

A screenshot of a cell phone

Description automatically generated

Fig 10.4

Error Rates

Following determination of the optimal hyperparameters for each combination of learner and dataset, I set up each classifier accordingly and trained them using the complete training set. For each dataset, the classifier performance is very similar. This gives me confidence the algorithms are performing near optimally.

Table 5: Classifier accuracy (%) using Titanic dataset.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training | Testing | Delta |
| Decision Tree | 88.0 | 84.4 | 3.6 |
| Multilayer Perceptron | 86.8 | 84.0 | 2.8 |
| Boosting | 86.4 | 84.0 | 2.4 |
| Support Vector Machine | 87.4 | 82.8 | 4.6 |
| K-Nearest Neighbors | 87.0 | 82.4 | 4.6 |

Table 6: Classifier accuracy (%) using Adult dataset.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Training | Testing | Delta |
| Decision Tree | 81.3 | 80.7 | 0.6 |
| Multilayer Perceptron | 80.8 | 80.8 | 0.0 |
| Boosting | 81.2 | 81.0 | 0.2 |
| Support Vector Machine | 79.8 | 79.4 | 0.4 |
| K-Nearest Neighbors | 81.3 | 80.9 | 0.4 |

The testing accuracies are within 2 and 1.6 percentage for the Titanic and Adult datasets, respectively. I find this relatively tight clustering of the classifier performance surprising. This dataset usability likely accounts for these datasets’ very high popularity among machine learning practitioners.

Training Time Analysis

Note that Table 7 uses milliseconds as the timescale, whereas Table 8 uses seconds.

Table 7: Training time analysis using Titanic dataset.

|  |  |
| --- | --- |
|  | Avg Training Time (ms) |
| Decision Tree | 2 |
| Multilayer Perceptron | 803 |
| Boosting | 8 |
| Support Vector Machine | 22 |
| K-Nearest Neighbors | 2 |

The training datasets differ in sample count by a factor of roughly 35 (36809 for Adult, 1047 for Titanic). The column count is higher for the Adult dataset as well, at 22 columns versus 15 for Titanic. The actual feature count that these columns encoded are similar at 7 and 8, respectively, however.

Table 8: Training time analysis using Adult dataset.

|  |  |
| --- | --- |
|  | Avg Training Time (s) |
| Decision Tree | 0.1 |
| Multilayer Perceptron | 10.9 |
| Boosting | 1.3 |
| Support Vector Machine | 53.1 |
| K-Nearest Neighbors | 0.5 |

Of interest, the time to train the neural network increased by a factor of roughly 14, whereas the time required to train the support vector machine increased by a factor of around 2400. The remaining classifiers increased in training time by factors intermediate between those two extremes.

Training Iterations Analysis

As discussed in the Hyperparameter Tuning section, I needed to increase the max\_iterations hyperparameter when using Stochastic Gradient Descent as my solver for the Titanic dataset. Iterating on the problem demonstrated an upper bound of 2000 iterations was sufficient. Ultimately, I used the “adam” solver, which converged after 500 or so iterations.

The Adult dataset required fewer iterations to train the model, presumably due to the much larger training sample set. Both the model set up for the Titanic dataset and the Adult dataset used the Logistic activation function and the “adam” solver. The only difference between them was the previously mentioned increased max iterations parameter and the size of the hidden layer, which was 12 and 17 neurons for Titanic and Adult, respectively. In summary:

• Titanic, 12 hidden neurons: 458 iterations

• Adult, 17 hidden neurons: 185 iterations

The relatively high iterations for Adult are accounted for by the more complex hidden layer, and the larger number of inputs (17 versus 24).

Conclusions

• Following hyperparameter tuning, all five classifiers exhibited strong performance on both datasets, with only slight overfitting on the smaller Titanic dataset.

• Performance on the Titanic dataset would likely be improved by training on a larger dataset. Practically, this could be accomplished by using a smaller test set (perhaps 10% rather than 20%).

• Decision tree, boosting, and the neural network each outperformed the other two classifiers on the Titanic dataset by a relatively significant 1.2% - 2.0% accuracy. This can be accounted for by how predictive passenger sex was in determining survival likelihood on the Titanic. The nature of those three classifiers is more conducive to heavily weighting that single feature than are the other two classifiers.

• The classifiers are listed below in order of decreasing “composite accuracy,” or the sum of both testing accuracy percentages. Note that the hyperparameters were optimized differently for the different datasets.

1. Decision Tree (165.1)

2. Boosting (165.0)

3. Multilayer Perceptron (164.8)

4. K-Nearest Neighbors (163.3)

5. Support Vector Machine (162.2)