Room Occupancy Classification

Ben Lyche
University of Oregon
951670389
blyche@uoregon.edu

Abstract

Determining the ways in which a room is being used can allow for energy savings and efficiency, especially on a large scale. Having the ability to accurately predict whether a room is occupied or not is a crucial step in determining what resources should be directed to the room. In this project I examine three different classification algorithms in order to solve this problem. The models being covered are k-nearest neighbors, logistic regression, and Naive Bayes. After tuning each of these model's hyperparameters to achieve the best possible accuracy, this project shows that k-nearest neighbors is the best performing model with an accuracy of 90.05% although all models covered have a similar and acceptable accuracy score.

1 Introduction

Properly predicting room occupancy status can be an excellent tool, especially for large office spaces looking to reduce energy costs. Determining when and how humans are using a space is a great way to mitigate energy expenditure, especially considering how much energy can be wasted with poorly optimized heating or lighting of rooms (Saralegui et al., 2019). With this in mind it is crucial to accurately and efficiently determine a way to make these predictions.

In this project I look to find the optimal machine learning algorithm for solving this problem. For the sake of the scale of this project and the data that will be used, I will treat this as a binary classification problem. This means my project will be using classifiers to determine one of two labels on the data: occupied or unoccupied.

This project will compare three different classifiers to ultimately determine the most efficient and accurate solution to this question of room occupancy. The first will be the k-nearest neighbors algorithm, the second being logistic regression, and the third being Naive Bayes. These three algorithms each offer different strengths and weaknesses and were chosen for this reason.

The remainder of the paper includes my background on the problem and algorithms in Section 2, my methodology in Section 3, the experiment results in Section 4, and finally, concluding thoughts in section Section 5.

2 Background

This problem entails correctly classifying room occupancy based on 4 features. The features are: temperature (in Celsius), humidity (measured in the standard fashion), CO2 content (in ppm), and humidity ratio (derived from temperature and relative humidity). With these four features I attempt to consistently and accurately predict a room's occupancy status. The data being used originally contained a fifth feature: light, however this feature mapped so closely with the label that the problem becomes much more interesting and informative when it is removed. For this reason, I proceed with the project without this fifth defining feature. Within the data there are 2,665 total examples, 972 being positive and 1,693 being negative. Although there are more negative examples than positive, with the amount of

total examples, this should not cause any issues in training the models.

The goal within this problem is discovering meaningful relationships between these four features that can ultimately help to determine a room's occupancy status. Next I will look to the methodology of the experiment, focusing on the classifiers that have been chosen and the reasoning behind them.

3 Methods

3.1 Classifiers

I will be comparing three different classifiers in this project. I will take a brief look at each of these algorithms. In order to implement all of these models I will be using the python library, scikit-learn.

3.1.1 K-nearest Neighbors

K-nearest neighbors is a well known classification algorithm. It can be a highly effective method of classification when the setting is right however there are areas where it can falter. In particular, in problems with a high dimensionality of features, this algorithm can suffer (Taunk et al., 2019). Fortunately, for this problem there is a very limited feature space which should lend itself well to this algorithm.

3.1.2 Logistic Regression

Logistic regression is another popular option for a boolean classification problem. Logistic regression uses probability scores and weights given to each feature in order to classify given examples. In general, what is seen with logistic regression is that, as the training set size increases, so too does the performance of the model at a rate that usually outpaces other classification models (Ng and Jordan, 2002).

3.1.3 Naive Bayes

Finally I will be comparing the other two to the Naive Bayes model. Similarly to logistic regression, Naive Bayes will be attributing a probability of an example being of a certain label. That said, Naive Bayes has been shown to generally reach a solution faster than models like linear regression (Ng and Jordan, 2002). Another key difference is the assumption of independence between features that comes with Naive Bayes. While this is usually never the case with data, Naive Bayes assumes a large degree of independence between the given features.

3.2 Tuning

In order to tune each model I will be utilizing a grid search method. Because of the smaller feature space and relatively simple models, it is feasible for this project to use grid search to optimally tune each of the models that will be tested.

Because scikit-learn will be used to implement the models, I will also use its grid search class in order to efficiently optimize during testing.

3.3 Features

In the case of this problem there is not much to say about the features. Because there is such a small feature space there will be no process of feature selection. I will be taking into account each of the features given.

To reiterate, I have removed the light feature as it almost perfectly matches the label and makes for a problem that is uninteresting, especially if one of the goals of this project is determining when lights should be turned on or off.

4 Experiments

4.0.1 Procedure

I begin by splitting the data into training and testing data. I use 80% as training data and 20% as testing data. That is 2,132 training examples and 533 testing examples. Note that a random state seed is used to keep the example consistent across all training for all models. This ensures fair and accurate comparison on this problem.

After making this split I follow the same general procedure for each model. This begins by doing some preliminary training on individual hyperparameters. This allows for the definition of viable ranges that possible optimal hyperparameters could fall within. Once these ranges have been found, I will use the grid search method to find the optimal solution for each model. In tuning the hyperparameters I will be striving for the best accuracy of the model. Finally I will compare the models with their optimally tuned hyperparameters and draw my conclusions from there.

4.0.2 K-nearest Neighbors

The most basic model with no tuning yields an accuracy of 87.24%. By adjusting some of the

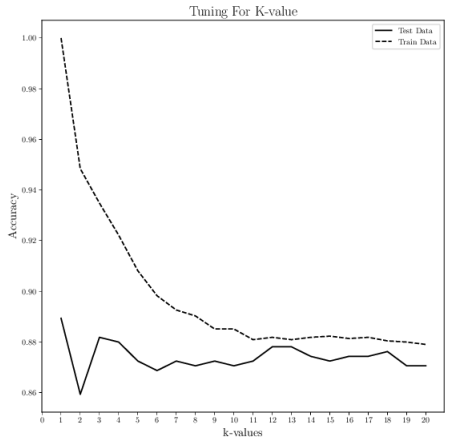

Figure 1: Visulaization for tuning of k-value

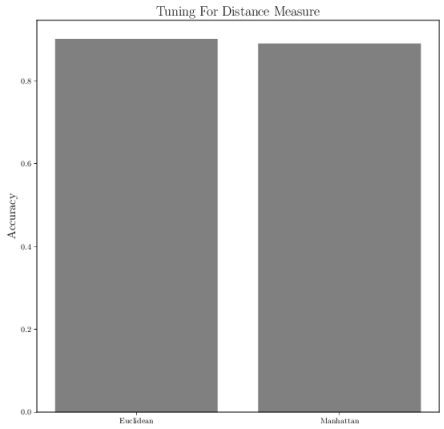

Figure 2: Visualization for tuning of distance measure

hyperparameter values, more performance can be squeezed out of the model.

I start tuning by looking at the most important parameter for k-nearest neighbors, the k-value itself. The results from this tuning can be found in Figure 1. From these results, it can be seen that I will want to check in the 1-18 range for these values of k. Anything outside of this range and it appears that the accuracy begins to fall off. That said, the accuracy is too unpredictable within this range to determine exactly what the k-value should be without more information on other hyperparameters.

The next important hyperparameter for us to examine on its own is the measure of distance that I will use for this model. For this project I will look at both a Euclidean measure of distance and Manhattan distance (essentially this is the sum of the distance in each dimension).

As seen in Figure 2, there is a minimal difference in accuracy for these hyperparameters however Euclidean distance outpaces Manhattan distance by about 2% points in accuracy.

Now I move on to grid search in order to completely tune the model. For grid search I will use a range of k-values from 1-18, try both Euclidean and Manhattan distances, and additionally, look at weights for the neighbors. I will test both uniform (all points weighted equally) and distance (closer points given a higher weight) for the model. These three hyperparameters and the values laid out make up the grid for this model.

Hyperparameter	**Value**
k-value	1
Weights	uniform
Distance Measure	Euclidean

Table 1: Optimal hyperparameters for k-nearest neighbor

Ultimately Table 1 gives the optimal hyperparameter values for this problem. When tuning to these hyperparameters there is an observed accuracy of 90.06%. This is an increase of almost 3% accuracy for the model.

4.0.3 Logistic Regression

For the base logistic regression model I achieved an accuracy of 87.80%. Again, first tuning individual hyperparameters then moving to a grid search with a new range, I will optimally tune this model. It should be noted that through all of this testing and tuning, the maximum number of epochs will be set to 500. In the testing done, anything more than this did not yield higher accuracy.

First I examine perhaps the most important of the hyperparameters, the penalty or the regularizer. There are two options here, l1 or l2. With the bar chart given in Figure 3 it can be seen that the l1 regularizer performs slightly better than l2 with all else being equal. I now look to the C value (the strength of the regularizer) this basically controls to what degree the model regulates in the process of coming to convergence. It should be noted that in

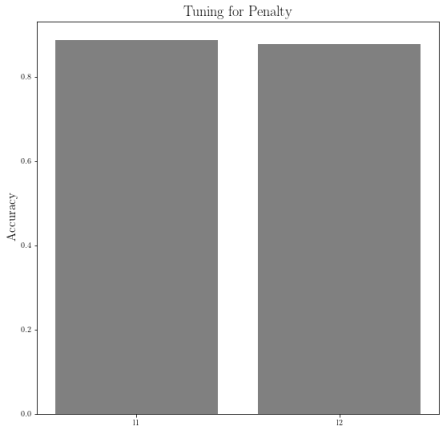

Figure 3: Visualization for tuning of penalty

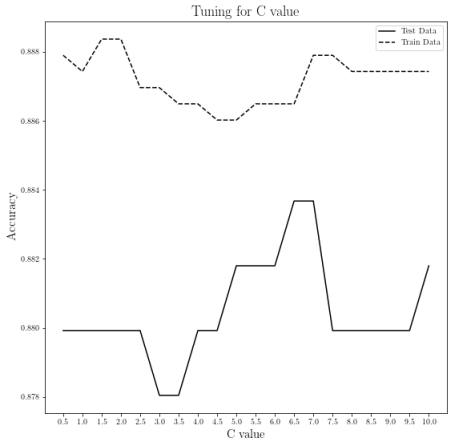

Figure 4: Visualization for tuning of C

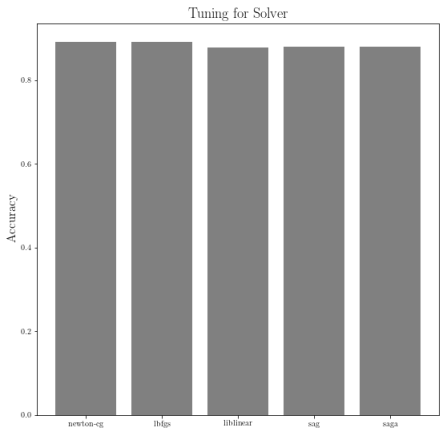

Figure 5: Visualization for tuning of solver

Hyperparameter	Value
C	1
Solver	newton-cg
Penalty	l2

Table 2: Optimal hyperparameters for logistic regression

Figure 4 I have set the penalty to l2 (the default) in order to continue the process of tuning each individual parameter first, then finishing with grid search. Figure 4 shows that there could lie an optimal value somewhere around the 6 or 7 range.

Lastly I compare the values of different solvers for the optimization problem. There are 5 different available solvers for this problem: newton-cg, lbfgs, liblinear, sag, and saga. In Figure 5 it is apparent that newton-cg and lbfgs have the best performance.

Ultimately there will be two grid searches that are run. One over C values in the range of 0-10 and the two penaltys l1 and l2 and the other over C values in the range of 0-10 and the 5 solvers. This is necessary because only the liblinear solver can handle the l1 penalty. The highest accuracy between these two optimized hyperparameters is taken. This accuracy comes out to 89.11% with the newton-cg solver, l2 penalty, and C = 1. This is a display of the necessity of grid search to find highest accuracy. The final C value ends up being much lower than what was originally found, likely because of the way it interacts with the newton-cg solver. Through tuning I have increased the accuracy of the model by almost 1% with the hyperparameters found in Table 2.

4.0.4 Naive Bayes

For Naive Bayes I start with a baseline accuracy of 86.30%. That said, the Naive Bayes model is a very simplistic one and there are not many hyperparametes to be tuned.

The hyperparameter I will be looking at is smoothing. In tuning this hyperparameter I seek to find the optimal portion to apply to variance smoothing. The values and the resulting accuracy is given in Figure 6. As can be seen, the optimal value lies at 1e-3.

This actually concludes the tuning for the Naive Bayes model, as mentioned previously, there are very few hyperparameters to actually tune. Thus I have achieved a final accuracy of 87.80% improving by over a full 1% from the original model.

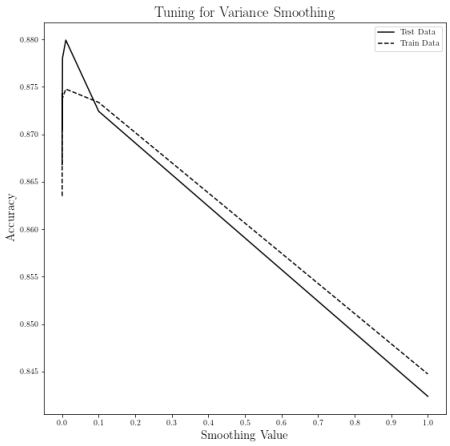

Figure 6: Visualization for tuning of smoothing

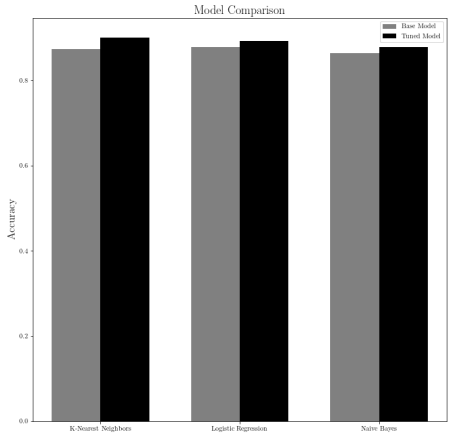

Figure 7: Comparisons for all models before and after tuning

4.0.5 Comparison

Now that I have finally fully tuned each of the models, I look to make a comparison between both the base models and their fully tuned counterparts. Comparisons will be made based on accuracy as this is the metric that has been used to train these models throughout the entire project.

In Table 3 the base and tuned model accuracies are compared. The table shows that the best performing model after tuning is the k-nearest neighbors model. It is also the model that shows the highest degree of improvement between the base and tuned models. Following the k-nearest neighbor model is the logistic regression model. The difference between these two models is very narrow and both show very good accuracy. Finally there is the Naive Bayes model with the worst accuracy. Naive Bayes, while not inaccurate, was outperformed by the other two models. Thus, the highest accuracy achieved in this project was 90.05% by k-nearest neighbors.

Model	Base Model Accuracy	Tuned Model Accuracy
K-nearest Neighbors	87.24%	90.05%
Logistic Regression	87.80%	89.12%
Naive Bayes	86.30%	87.80%

Table 3: Comparison of test accuracy for base models and tuned models

It is interesting to observe that the k-nearest neighbors algorithm also preferred simplicity in its hyperparameters. The model performing best when only looking at the closest neighbor with a Euclidean measurement of distance and no extra weight given to each point.

It is clear that all models still performed well for this problem and the difference between them is minimal. That said, I believe that k-nearest neighbors has outperformed the other two models due to the small nature of the number features and the simplicity of this problem. This is an area where the k-nearest neighbor algorithm excels. All things being equal k-nearest neighbors would be the model to choose for a problem of this style, however the accuracies are close enough that both other models should not be dropped out of consideration immediately. With more features it is likely that the logistic regression model would surpass the k-nearest neighbors.

5 Conclusion

On this project I have compared three different machine learning algorithms and their ability to accurately solve the problem of determining if a room is occupied or not. Using these three algorithms I have achieved high but varying degrees of accuracy in solving this problem. Having tuned the hyperparameters, the best performing model was the k-nearest neighbors model, likely due to the simplicity of this problem and its features.

In conclusion, this is only the surface of this problem. This project gives an idea as to some models that could be useful in making these predictions but

much more could be done once the predictions have been made. For example, looking into the degree of use a room is receiving and then determining what kind of resources need to go to this room (i.e. heating/cooling, lighting, etc.). This could look like determining how many people are in a room rather than whether it is simply occupied or not. This project suggests some models that could be considered if delving further into this subject.

References

Andrew Y. Ng and Michael I. Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*, pages 841–848.

Unai Saralegui, Miguel Angel Anton, Olatz Arbelaitz, and Javier Muguerza. 2019. Smart meeting room usage information and prediction by modelling occupancy profiles. *Sensors*.

Kashvi Taunk, Sanjukta De, Srishti Verma, and Aleena Swetapadma. 2019. A brief review of nearest neighbor algorithm for learning and classification. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1255–1260.