Hyperparameter Tuned Classifiers

Evan Phoukong
University of the Pacific
Stockton, California
e_phoukong@u.pacific.edu

***Abstract*—The objective of this analysis was to determine the best model that could differentiate the desired components in a thermal image. To do this, a multitude of classifiers were configured, including K-Nearest Neighbors, Naive Bayes, Decision Trees, Random Forests, and Support Vector Machines, utilizing as many variations of their respective parameters to achieve the best results. Hyperparameter tuning was neccessary to ensure that each of these models where most optimal for the data.**

I. Introduction

A model that could classify various hot spots in a thermal image would be very beneficial to our primary stakeholder, the CA Fire Department. They would be able to use this model to recognize thermal abnormalities in an image and address them accordingly. Utilizing the capabilities of supervised machine learning, a model could be trained on various thermal images to automatically identify various hot spots and determine what they are. Hyperparameter tuning is essential when developing the models to ensure that they could best fit the data. A model tailored towards this specific data set would be able to achieve higher accuracy and a reduction in mis classifications when identifying hot spots.

The first step in creating this model was to collect and label the data that would be used to train it. The data was sourced from a publicly available thermal image data set [1], and a script was run on each image to create a separate binary image that distinguished hot spots. Labeling of these hot spots as well as their estimated temperature was done manually for all images, and their results were saved into a number of CSVs that were then unified into one data set.

Exploratory data analysis was performed on this data set before it was passed to the models. Missing values, NaNs, and duplicate rows were all dropped as they offered no value to the classification process. A significant challenge encountered during this process was handling human error during the data collection process. Improper labeling seemed to be the most prevalent issue in the case. To account for improper labeling, all values in numerical columns were checked to ensure they where numeric and where replaced with NaN if they were not. For the ”Class” column, the various categories were checked to make sure that they aligned with expected values and were replaced with NaN otherwise. Empty values or missing values where also classified as NaN and ultimately dropped. Finally, the ”filename” column was dropped as it served as an identity number/name for each image and offered no intrinsic value to the models.

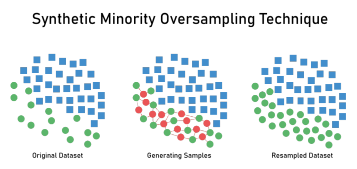

Fig. 1. An example of SMOTE synthetically created more observations (represented in red) for the minority class,represented in green) .

After curating the data, the next step was identifying the target variable and features. The target variable in this case was ”Class”, with values encompassing ”tire” and ”car engine” (for thermal images pertaining to cars”, and ”head”, ”torso”, ”arm” and ”leg” (for thermal images pertaining to people. ”Noise” was the last possible value that was given to hot spots that did not fall under the categorical values listed previously. Since the target variable is categorical, it had to be encoded into numerical values before being passed tot he models. Feature variables included ”temperature”, ”xcenter”, ”ycenter”, ”eccentricity”, ”area”, ”angle”, ”perimeter”, ”hue” and ”saturation”. Finally, a correlation matrix was implemented to determine what features best correlated with the ”Class” variable, and those with a correlative value greater than 0.65 in magnitude where used.

Another major challenge during the data analysis process was class imbalance. For instance, the largest label, ”noise” was observed 1677 times, about 2 times more than the second most observed label, ”torso” (at 810), and about 11 times larger than the smallest observed label, ”tire” (at 154). This huge disparity in the data would be detrimental when training the models since they simply would not have enough data to categorize each hot spot accurately. To counteract this, a technique called Synthetic Minority Oversampling, or SMOTE, was implemented (Refer to Fig. 1). How SMOTE works is that it artificially creates minority class values, such as ”tire” mentioned previously, utilizing already existing observations to mimic what an actual observation would appear like in the data. Employing this technique resulted in all classes being evenly distributed, with 1677 observations in each category. It’s important to recognize that the SMOTE technique was not employed for data being given to the Naive Bayes Classifier. This is because Naive Bayes uses probability rather than distance as its metric when classifying values, and SMOTE would only impede the probabilistic nature of the classifier.

II. Classifiers

A. K-Nearest Neighbors

K-Nearest Neighbors, or KNN, is a model that uses distance to classify new observations. Typically, euclidean or manhattan metric are used when calculating the distance. The observation being classified is compared to its K nearest feature points, where K is a number determined by the engineer implementing the model. K should be an odd number to ensure an uneven split among the classes and guarantee a majority class. The majority class among the neighbors is then assigned to the new observation. KNN is able to benefit from the SMOTE technique mentioned previously since its primary metric of evaluation is distance. Having more neighbors for each label within the class means more accurate fitting and classification. Another big advantage of KNN is its versatility when setting the neighbors to consider. More neighbors can be more accurate classification, but this introduces a substantial downside to this model which is it being computationally expensive. Find the nearest neighbors and computing their distances is very time consuming since you would have to iterate through all the data points.

B. Support Vector Machines

Support Vector Machines, or SVM, is a model that classifies observations based on what side of a kernel they fall on. A kernel is a hyperplane that attempts to evenly separate classes. An example to help visualize this would be the kernel for a class containing two possible values (such as ”Yes” or ”No”). The kernel in a 2 dimensional space would be a line that does it best to evenly divide ”yes” observations and ”no” observations. When dealing with more than two dimensions, this kernel becomes a hyperplane. Data points that fall near the kernel are support vectors, and the distance between the closest support vector and the kernel is the margin. A big advantage of SVMs are their ability to handle nonbinary classes by classifying them using hyperplanes. They are also robust to outliers since the kernel takes into account the general relationship of different classes. SVM also has a plethora of adjustment options that allow it to handle unconventional data. For instance, the engineer implementing the model can perform a kernel trick which involves increasing the order of the kernel, typically for data that has a non-linear relationship. Engineers can choose how much they would like to regularize the SVM. This essentially allows the engineer to determine how general or how specific to the given data they want their model to be, and the regularization parameter will be discussed in more detail later. There are some drawbacks to the SVM. For instance, SVM is computationally expensive due to the distance calcualtions, and some data may not have a clear relationship that could fit well with any kernel. Depending on how you set the regularization parameter, the model may be prone to either classifications or overfitting, and although these two consequences are balanced by an increase in margin maximization and accuracy, respectively, it is still something to consider when trying to optimize SVM.

C. Naive Bayes

Naive Bayes is a simple but effective model that uses the Bayes Theorem to classify different observations, using probabilities pertaining to different classes. It makes the naive assumption that all the features are independent of one another and, given these feature points, it calculates the probability of a specific class. There are many different varieties of Naive Bayes, and the one implemented in this project was Guassian Naive Bayes. Guassian NB works with continuous features and forms a Gaussian distribution, which can be used to calculate the probability of a specific class. Its advantageous in the fact it’s a simple, computationally efficient model that does not fall victim to class imbalances due to its probabilistic nature. It also works well when features are necessarily numerical. A major drawback of this model is that its performance and stability are highly dependent on the naive assumption being true. If the assumption is false, its accuracy and reliability take a drastic hit.

D. Decision Trees

Decision tree models work by choosing the best features to split and repeating this process for each feature until there is none left or no splitting is possible. The best attribute is determined using an ID3 algorithm that seeks to minimize a metric, usually entropy (measures of disinformation), with each split. With each split, a question from the parent node is answered and new question is asked in the child node. This way, given different feature values, the tree will follow differnt paths and try to pick the class that best fits a new observation. The advantages of decision trees are that its robust to outliers and simple/versatile for data with non-linear relationships. A major drawback is that it is susceptible to overclassification and is biased toward feature values with a larger amount of different values.

E. Random Forests

Random Forests is a model that implements multiple decision trees to make it’s decision. It divides the given data set into subsets that it passes to each tree. Then, it takes the result of each decision tree and averages them out to determine what class to assign to new observation. Since it is implementing multiple decision trees, it is far more accurate and capable of handling data that has an imbalance or missing values. It is however, very computationally effective since creates multiple decision trees, and it relies on their being enough data to pass on to the different trees.

III. Hyperparameter Tuning

KNN	**SVM**	**Naive Bayes**	**Decision Tree**	**Random Forest**
n_neighbors	C	var_smoothing	criterion	n_estimators
weights	kernel		max_depth	max_depth
metric	gamma		min_samples_split	min_samples_split
			min_samples_leaf	

TABLE I
Hyperparameters for Classifiers

A. *Hyperparameters for KNN*

The ”n_neighbors” sets the K value for KNN, and determines the amount of neighbors to consider when classifying a point. The numbers 3, 5, 7, and 9 where inputted for this parameter, with 3 yielding the best result. This may be due to the fact that it’s less susceptible to over fitting, and since SMOTE was used to fix imbalance issues with the class, the issue of bias to a specific class was less of an issue. The ”weights” parameter determines how the distances are used to classify different observations, and the values ”uniform” and ”distance” were passed to it. ”Uniform” means the closest K distances are weighted equally, and the majority class will solely determine the classification. ”Distance” means each distance is weighted, and a neighbor (feature point) closer to the observation will have higher value when classifying then those farther from it. ”Distance” yielded the best results in this project. Finally, ”metric” is used to determine how distance is calculated and what metric is used. ”Euclidean”, ”Manhattan”, and ”Minkowski” were the three metrics used, with ”Manhattan” yielding the best results.

B. *Hyperparameters for SVM*

The ”C” parameter is the regularization parameter that controls how much the model values margin maximization/misclassifications vs accuracy/overfitting. It’s typically given logarithmic values as input, and in this project used 0.1 and 1, with 1 yielding the best result since this told the SVM to favor accuracy more. The ”kernel” parameter specifies the kernel type. ”Linear” and ”RBF” were passed to this parameter. ”Linear” sets the dimensions of the input to be a line and are used, as the name suggests, for linear data. ”RBF” makes the kernel a higher order and polynomial in nature to represent data with nonlinear relationships. Since the data was more than two dimensions and definitely not linear in nature, ”RBF” yielded the best results. Finally, the ”gamma” parameter controls how specifically tuned the engineer what the model to be to a specific data set. ”scale” and 0.1 were passed as the input, with scale using the best results since it dynamically determines, based on the data set, how much influence it has.

C. *Hyperparameters for Naive Bayes*

The only hyperparameter considered for Naive Bayes was ”var_smoothing”. This a set number number added to all variances using in calculation to prevent divide by zero errors and ensure stability. The values inputted where 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, with 1e-7 yielding the best results.

D. *Hyperparameters for Decision Tree*

The hyperparameter ”criterion” determines the metric that trees use to determine how t split nodes. ”Gini” and ”entropy” can be passed, with each split trying to minimize this number. ”Entropy” yielded the best result in this case since the metric was more applicable to the data. ”Max_depth”, as indicated by the name, determines the max depth of tree. If you have enough computation power, not having a max depth is the best since the tree can perform as many calculations and splits as necessary. This is why ”none” was the best input for this parameter. The ”min_samples_split” determines the number of samples that are needed for a node to be able to split, and ”2”, ”5”, and ”10” were inputted into this parameter.. Preferably, you want your tree to make as many splits as possible for more accurate results, and that is why ”2” yielded the best results. Finally, ”min_samples_leaf” determines the minimum samples needed for a node to be considered a leaf. The values inputted where ”1”, ”2”, and ”5” were inputted into this parameter, with ”1” yielding the best result since a leaf node with 1 sample would not be able to split further.

E. *Hyperparameters for Random Forest*

The hyperparameter ”n_estimators” sets how many decision tree should be created, and ”50” and ”100” were passed to this with ”100” performing better. A higher number leads to better accuracy, but more computational power and time. ”Max_depth” set the maximum depth of the decision tree, and ”None” and ”10” were passed to this parameter. The only reason one should set a limit, even though it decreases accuracy and performance, is if computational power is limited, and ”None” will always give the most accurate results. Finally, ”min_samples_split” set the minimum number of samples needed for a node to split, and a lesser number means more splits and better accuracy. ”2” and ”5” were passed to this parameter with ”2” performing better.

IV. Results

KNN scored 0.85 for accuracy, precision, recall, and F1, and its confusion matrix can be referenced in Fig 2. KNN overall was very accurate across all charts, and it classified an equal amount of false positives as it did false negatives. It was likely to perform well since SMOTE was implemented and there was a large dataset.

SVM scored 0.66 for accuracy, precision, recall, and F1, and its confusion matrix can be referenced in Fig 3. SVM was not as accurate as KNN, but it was similar in that it made an equal amount of false positives and negatives. There likely wasn’t an rbf kernel that could fit the data and all 7 classes.

Naive Bayes scored 0.47 for accuracy and recall, .42 for precision, and .39 F1, and its confusion matrix can be referenced in Fig 4. Since the data had a an exceptionally large class imbalance, Naive Bayes just simply didn’t have enough data in other categories to perform accurate classifications. Due to its probabilistic nature, SMOTE would not benefit it.

Decision Tree scored 0.79 for accuracy, precision, recall, and F1, and its confusion matrix can be referenced in Fig 5. Since computational power didn’t really limit how large the tree could be, the tree was allowed to be quite extensive and therefore very accurate.

Random Forest scored 0.88 for accuracy, precision, recall, and F1, and its confusion matrix can be referenced in Fig 8. Since computational power didn’t really limit how many tree it could make, Random Forest could make an average the results of 100 tree, which allowed it to have very high score across all charts.

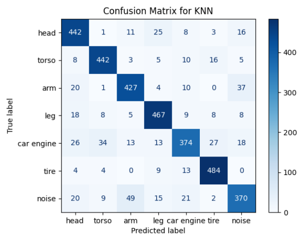

Fig. 2. Confusion Matrix for KNN

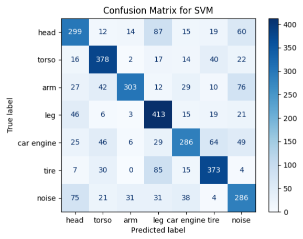

Fig. 3. Confusion Matrix for SVM

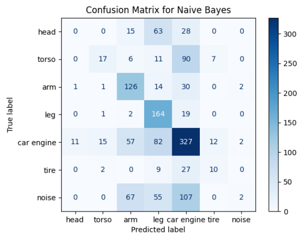

Fig. 4. Confusion Matrix for Naive Bayes

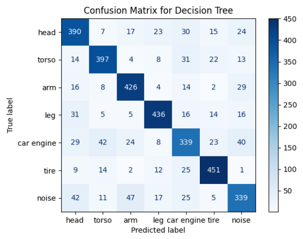

Fig. 5. Confusion Matrix for Decision Tree

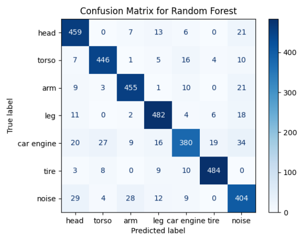

Fig. 6. Confusion Matrix for Random Forest

V. Conclusion

It is evident that the Random Forest classifier is the most suitable for classifying thermal anomalies/hot spots. It scored the highest in all categories, boasting a 0.88 score in accuracy, precision, recall, and F1-score. Observing the confusion matrix, the high numbers and dark blue colors along the diagonal indicate it was very accurate in classifying all classes. Even if computational power is a limiting factor, one could still reduce the number of trees created to achieve a still high accuracy score. The Decision Tree classifier had an accuracy of 0.79, and this shows the relative minimum that Random Forests can score if "n_estimators" was set to one.