Kaggle Miniproject Report

Alvita Tran

Overview

My project consisted of an investigation into the performance of several common machine learning methods, as implemented in Python's scikit-learn toolbox. After finding the best parameters for each model and the validation accuracy rates they individually achieved, I then looked into the performance of ensembles that classified examples based on the majority vote of different combinations of the individual methods. I tested bagging and boosting ensembles with different base estimators as well. Finally, I also learned about the effects of different feature representations of the provided dataset on models' scores.

Data Manipulation

I tried a couple separate manipulations of the data from its raw document-word form. The first was dimensionality reduction using Latent Dirichlet Allocation, with the rationale that representing documents as mixtures of topics is sometimes more informative and effective than using word count vectors. I tested the performance of various simple classifiers using the representation generated from Latent Dirichlet Allocation with n topics, for $n \in [5, 10, 20, 50, 100]$.

A different manipulation was tranformation of the count matrix into its tf-idf representation. A common treatment of text data, tf-idf considers both the frequency of a given word in a document and the frequency of the word in the corpus for weighting. I used sublinear tf scaling ($1 + \log$ tf), and normalized the matrix using the L2 norm.

The best results of a few simple classifiers (best as selected by cross validation, as described in Section 4) are shown in Table 1 for each of the different feature representations. It can be seen that the transformation derived from Latent Dirichlet Allocation does not provide a good representation of the data for this classification task; for all tested values of number of topics, training models on the reduced representation resulted in poorer scores than training on the raw feature matrix or on the tf-idf representation. Using tf-idf resulted in accuracy that was comparable to or better than that achieved by using the original representation. For this reason, I proceeded with the tf-idf-transformed data for my learning algorithms.

	5 Topics	10 Topics	20 Topics	50 Topics	100 Topics	tf-idf	Original
Logistic Regression	0.572	0.568	0.625	0.621	0.638	0.689	**0.694**
SVM	0.573	0.562	0.610	0.617	0.627	0.687	**0.690**
Decision Tree	0.595	0.570	0.594	0.614	0.640	0.646	**0.648**
Perceptron	0.527	0.536	0.539	0.591	0.589	**0.663**	0.648
K Nearest Neighbors	0.572	0.587	0.610	0.626	0.641	**0.672**	0.636

Table 1: Feature Representations

Learning Algorithm

As an exploration of different model classes, I tested the following supervised learning algorithms. Reported in Table 2 are their best validation accuracy rates, as well as abbreviations for simplicity in following discussion.

G-NB	Gaussian Naive Bayes	0.654
M-NB	Multinomial Naive Bayes	0.690
LR	Logistic Regression	0.689
Perc	Perceptron	0.663
Ridge	Ridge Classifier	**0.698**
SVM	Support Vector Machine	0.687
DT	Decision Tree	0.646
K-NN	K Nearest Neighbors	0.672
Rocchio	Nearest Centroid (Rocchio's Algorithm)	0.583
AB	AdaBoost	0.660
Bag	Bagging	0.663
GB	Gradient Boosting	0.691
RF	Random Forest	0.676
LDA	Linear Discriminant Analysis	0.666
QDA	Quadratic Discriminant Analysis	0.537

Table 2: Supervised Learning Algorithms

Based on these results, I tried a couple of highest scoring models – LR, SVM, and Ridge – individually on the test data, guided by the recommendation that linear models or large-margin classifiers often achieve strong performance on classfication of bags of words, and by the preference for a simpler model over a more complex one if it can sufficiently separate the data. Alone, these classifiers achieved scores in the range of $[0.618, 0.644]$ on the test data, across the public and private sets.

Since the simple models did not sufficiently separate the data, I was interested in improving performance by combining the predictions of these classifiers. With the 15 algorithms listed above, I iterated through the $\binom{15}{n}$ combinations of individual models, for $n \in [3, 5, 7, 9, 11, 13]$. It can be noted that some methods included in the list of 15 are already ensembles, namely boosting, bagging, and random forest algorithms. These "individual" methods are themselves composed of many weak learners, but I consider each one's final prediction as the output of a single algorithm for this experiment.

For each n, I found the subset of n classifiers that together, by hard majority voting, achieved highest validation accuracy. I expected that within each best combination, the selected classifiers would tend to be diverse in order to compensate for the different biases of each model class. This appeared to be true, and the ensemble learners also tended to be selected often. Repeating the experiment without the ensemble learners and with more individual learners of the linear model formulation (Bayesian Ridge, Elastic Net, Lars, Lasso, Orthogonal Matching Pursuit, and Passive Aggressive Classifier), resulted in overall lower validation accuracy rates, not reported. The best combinations of the reported 15 for each n are listed in Table 3.

n	Accuracy	Algorithms
1	0.698	Ridge
3	0.722	K-NN, GB, RF
5	0.729	LR, K-NN, Bag, GB, RF
7	0.729	LR, DT, K-NN, AB, GB, RF, LDA
9	**0.732**	LR, K-NN, Rocchio, AB, Bag, GB, RF, LDA, QDA
11	0.726	LR, SVM, DT, K-NN, Rocchio, AB, Bag, GB, RF, LDA, QDA
13	0.718	G-NB, LR, Perc, Ridge, DT, K-NN, Rocchio, AB, Bag, GB, RF, LDA, QDA
15	0.698	G-NB, M-NB, LR, Perc, Ridge, SVM, DT, K-NN, Rocchio, AB, Bag, GB, RF, LDA, QDA

Table 3: Model Combinations

On the test data, ensembles of sizes 5, 7, 9, and 11 achieved accuracy rates in $[0.632, 0.698]$, improving performance over the individual algorithms alone. As on the validation set, the ensemble of 9 algorithms earned the highest score. For a final test, I also tried bagging and boosting ensembles with different base learners (M-NB, LR, Perc, DT) for different numbers of estimators. These ensembles tended to overfit; Figure 1 demonstrates that for a bagging classifier with decision tree weak learners, training error $\ll$ validation error, and the same qualitative relationship was true when using other base learners as well. The best bagging classifier scored 0.661 on the test data, lower than the best manual ensemble classifier's score. In summary, ensembles of different methods were more successful than ensembles of the same method, even when the number of estimators was optimized for the latter.

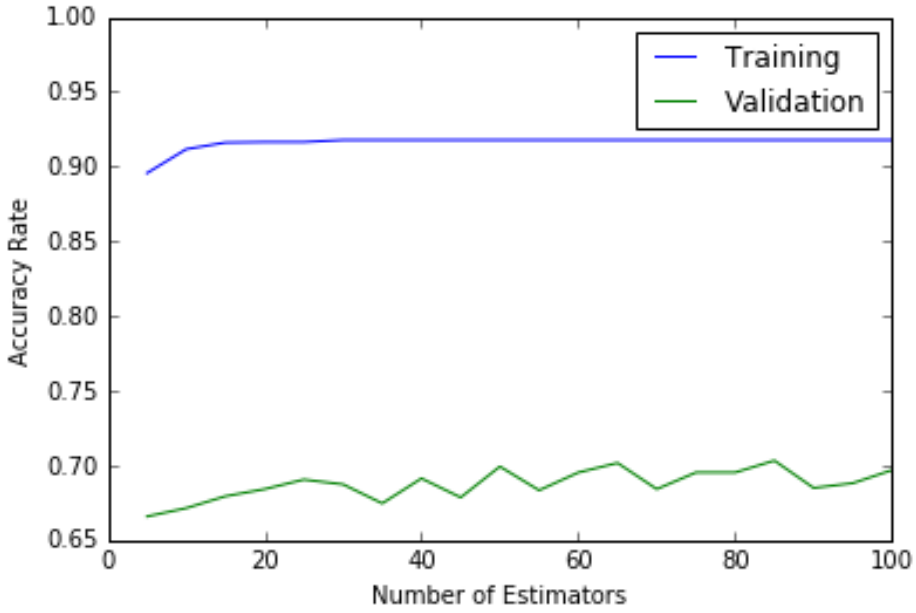

Figure 1: Bagging Ensemble Classifier

Model Selection

I used 70% of the training data as a training set and the other 30% as a validation set. For each algorithm, I trained models with different parameter values using the training data, and choose the best parameter, or parameter combination, as that which produced the highest accuracy on the validation set. Below, for applicable methods I report the parameter(s) I tuned and the range of values I tried.

- M-NB: alpha $\in [10^{-4:4}]$
- LR: C $\in [10^{-4:4}]$
- Perc: alpha $\in [10^{-4:4}]$
- Ridge: alpha $\in [10^{-4:4}]$
- SVM: C $\in [10^{-4:4}]$, kernel $\in$ ['linear', 'rbf', 'polynomial', 'sigmoid']
- DT: min_samples_leaf $\in [1:30]$, max_depth $\in [4:30]$
- K-NN: n_neighbors $\in [3:2:15]$
- AB: n_estimators $\in [20:10:200]$
- Bag: n_estimators $\in [5:5:100]$
- GB: learning_rate $\in [10^{-4:4}]$
- RF: n_estimators $\in [5:5:100]$

Conclusion

Through my project, I experimented with different feature transformations, model classes, and ensemble methods. I achieved the best performance with ensembles of diverse learning algorithms, and with more time would have been interested in seeing if ensemble accuracy would further improve with the addition of more model classes, such as neural networks, or saturate.