LATIHAN FINAL

MATA KULIAH	: Kecerdasan Mesin (IN242)	TANGGAL RILIS	: 7 Juni 2022
SEMESTER	: Genap 2021/2022	WAKTU	: Tengah Malam
DOSEN	: HB & HT	POIN	: 100

Problem 1. What is the fundamental idea behind Support Vector Machines?

Jawab: The fundamental idea behind Support Vector Machines is to fit the widest possible ”street” between the classes. In other words, the goal is to have the largest possible margin between the decision boundary that separates the two classes and the training instances. When performing soft margin classification, the SVM searches for a compromise between perfectly separating the two classes and having the widest possible street (i.e., a few instances may end up on the street). Another key idea is to use kernels when training on nonlinear datasets.

Problem 2. What is a support vector?

Jawab: After training an SVM, a support vector is any instance located on the ”street” (see the previous answer), including its border. The decision boundary is entirely determined by the support vectors. Any instance that is not a support vector (i.e., is off the street) has no influence whatsoever; you could remove them, add more instances, or move them around, and as long as they stay off the street they won’t affect the decision boundary. Computing the predictions only involves the support vectors, not the whole training set.

Problem 3. Why is it important to scale the inputs when using SVMs?

Jawab: SVMs try to fit the largest possible ”street” between the classes (see the first answer), so if the training set is not scaled, the SVM will tend to neglect small features (see Figure 1).

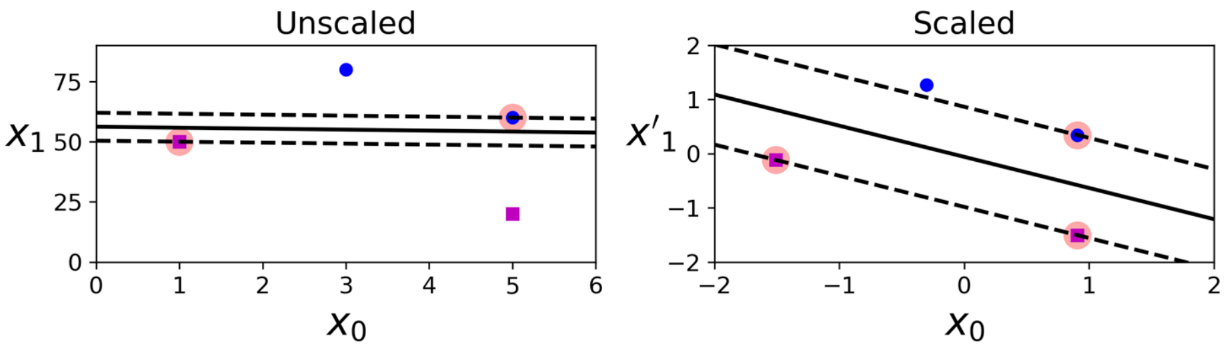

Figure 1: Sensitivity to feature scales

Problem 4. Can an SVM classifier output a confidence score when it classifies an instance? What about a probability?

Jawab: An SVM classifier can output the distance between the test instance and the decision boundary, and you can use this as a confidence score. However, this score cannot be directly converted into an estimation of the class probability. If you set `probability=True` when creating an SVM in Scikit-Learn, then after training it will calibrate the probabilities using Logistic Regression on the SVM’s scores (trained by an additional five-fold cross-validation on the training data). This will add the `predict_proba()` and `predict_log_proba()` methods to the SVM.

Problem 5. Say you've trained an SVM classifier with an RBF kernel, but it seems to underfit the training set. Should you increase or decrease γ (`gamma`)? What about `C` ?

Jawab: If an SVM classifier trained with an RBF kernel underfits the training set, there might be too much regularization. To decrease it, you need to increase `gamma` or `C` (or both).

Problem 6. What is the approximate depth of a Decision Tree trained (without restrictions) on a training set with one million instances?

Jawab: The depth of a well-balanced binary tree containing m leaves is equal to $\log_2(m)$, rounded up. A binary Decision Tree (one that makes only binary decisions, as is the case with all trees in Scikit-Learn) will end up more or less well balanced at the end of training, with one leaf per training instance if it is trained without restrictions. Thus, if the training set contains one million instances, the Decision Tree will have a depth of $\log_2(10^6) \approx 20$ (actually a bit more since the tree will generally not be perfectly well balanced).

Problem 7. Is a node's Gini impurity generally lower or greater than its parent's? Is it generally lower/greater, or always lower/greater?

Jawab: A node's Gini impurity is generally lower than its parent's. This is due to the CART training algorithm's cost function, which splits each node in a way that minimizes the weighted sum of its children's Gini impurities. However, it is possible for a node to have a higher Gini impurity than its parent, as long as this increase is more than compensated for by a decrease in the other child's impurity. For example, consider a node containing four instances of class A and one of class B. Its Gini impurity is $1 - \left(\frac{1}{5}\right)^2 - \left(\frac{4}{5}\right)^2 = 0.32$. Now suppose the dataset is one-dimensional and the instances are lined up in the following order: A, B, A, A, A. You can verify that the algorithm will split this node after the second instance, producing one child node with instances A, B, and the other child node with instances A, A, A. The first child node's Gini impurity is $1 - \left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2 = 0.5$, which is higher than its parent's. This is compensated for by the fact that the other node is pure, so its overall weighted Gini impurity is $\frac{2}{5} \times 0.5 + \frac{3}{5} \times 0 = 0.2$, which is lower than the parent's Gini impurity.

Problem 8. If a Decision Tree is overfitting the training set, is it a good idea to try decreasing `max_depth`?

Jawab: If a Decision Tree is overfitting the training set, it may be a good idea to decrease `max_depth`, since this will constrain the model, regularizing it.

Problem 9. If a Decision Tree is underfitting the training set, is it a good idea to try scaling the input features?

Jawab: Decision Trees don't care whether or not the training data is scaled or centered; that's one of the nice things about them. So if a Decision Tree underfits the training set, scaling the input features will just be a waste of time.

Problem 10. If it takes one hour to train a Decision Tree on a training set containing 1 million instances, roughly how much time will it take to train another Decision Tree on a training set containing 10 million instances?

Jawab: The computational complexity of training a Decision Tree is $O(n \times m \log(m))$. So if you multiply the training set size by 10, the training time will be multiplied by $K = (n \times 10m \times \log(10m))/(n \times m \times \log(m)) = 10 \times \log(10m)/\log(m)$. If $m = 10^6$, then $K \approx 11.7$, so you can expect the training time to be roughly 11.7 hours.

Problem 11. If your training set contains 100,000 instances, will setting `presort=True` speed up training?

Jawab: Presorting the training set speeds up training only if the dataset is smaller than a few thousand instances. If it contains 100,000 instances, setting `presort=True` will considerably slow down training.

Problem 12. If you have trained five different models on the exact same training data, and they all achieve 95% precision, is there any chance that you can combine these models to get better results? If so, how? If not, why?

Jawab: If you have trained five different models and they all achieve 95% precision, you can try combining them into a voting ensemble, which will often give you even better results. It works better if the models are very different (e.g., an SVM classifier, a Decision Tree classifier, a Logistic Regression classifier, and so on). It is even better if they are trained on different training instances (that's the whole point of bagging and pasting ensembles), but if not this will still be effective as long as the models are very different.

Problem 13. What is the difference between hard and soft voting classifiers?

Jawab: A hard voting classifier just counts the votes of each classifier in the ensemble and picks the class that gets the most votes. A soft voting classifier computes the average estimated class probability for each class and picks the class with the highest probability. This gives high-confidence votes more weight and often performs better, but it works only if every classifier is able to estimate class probabilities (e.g., for the SVM classifiers in Scikit-Learn you must set `probability=True`).

Problem 14. Is it possible to speed up training of a bagging ensemble by distributing it across multiple servers? What about pasting ensembles, boosting ensembles, Random Forests, or stacking ensembles?

Jawab: It is quite possible to speed up training of a bagging ensemble by distributing it across multiple servers, since each predictor in the ensemble is independent of the others. The same goes for pasting ensembles and Random Forests, for the same reason. However, each predictor in a boosting ensemble is built based on the previous predictor, so training is necessarily sequential, and you will not gain anything by distributing training across multiple servers. Regarding stacking ensembles, all the predictors in a given layer are independent of each other, so they can be trained in parallel on multiple servers. However, the predictors in one layer can only be trained after the predictors in the previous layer have all been trained.

Problem 15. What is the benefit of out-of-bag evaluation?

Jawab: With out-of-bag evaluation, each predictor in a bagging ensemble is evaluated using instances that it was not trained on (they were held out). This makes it possible to have a fairly unbiased evaluation of the ensemble without the need for an additional validation set. Thus, you have more instances available for training, and your ensemble can perform slightly better.

Problem 16. What makes Extra-Trees more random than regular Random Forests? How can this extra randomness help? Are Extra-Trees slower or faster than regular Random Forests?

Jawab: When you are growing a tree in a Random Forest, only a random subset of the features is considered for splitting at each node. This is true as well for Extra-Trees, but they go one step further: rather than searching for the best possible thresholds, like regular Decision Trees do, they use random thresholds for each feature. This extra randomness acts like a form of regularization: if a Random Forest overfits the training data, Extra-Trees might perform better. Moreover, since Extra-Trees don't search for the best possible thresholds, they are much faster to train than Random Forests. However, they are neither faster nor slower than Random Forests when making predictions.

Problem 17. If your AdaBoost ensemble underfits the training data, which hyperparameters should you tweak and how?

Jawab: If your AdaBoost ensemble underfits the training data, you can try increasing the number of estimators or reducing the regularization hyperparameters of the base estimator. You may also try slightly increasing the learning rate.

Problem 18. If your Gradient Boosting ensemble overfits the training set, should you increase or decrease the learning rate?

Jawab: If your Gradient Boosting ensemble overfits the training set, you should try decreasing the learning rate. You could also use early stopping to find the right number of predictors (you probably have too many).

Problem 19. What are the main motivations for reducing a dataset's dimensionality? What are the main drawbacks?

Jawab: The main motivations for dimensionality reduction are:

- To speed up a subsequent training algorithm (in some cases it may even remove noise and redundant features, making the training algorithm perform better)
- To visualize the data and gain insights on the most important features
- To save space (compression)

The main drawbacks are:

- Some information is lost, possibly degrading the performance of subsequent training algorithms.
- It can be computationally intensive.
- It adds some complexity to your Machine Learning pipelines.
- Transformed features are often hard to interpret.

Problem 20. What is the curse of dimensionality?

Jawab: The curse of dimensionality refers to the fact that many problems that do not exist in low-dimensional space arise in high-dimensional space. In Machine Learning, one common manifestation is the fact that randomly sampled high-dimensional vectors are generally very sparse, increasing the risk of overfitting and making it very difficult to identify patterns in the data without having plenty of training data.

Problem 21. Once a dataset's dimensionality has been reduced, is it possible to reverse the operation? If so, how? If not, why?

Jawab: Once a dataset's dimensionality has been reduced using one of the algorithms we discussed, it is almost always impossible to perfectly reverse the operation, because some information gets lost during dimensionality reduction. Moreover, while some algorithms (such as PCA) have a simple reverse transformation procedure that can reconstruct a dataset relatively similar to the original, other algorithms (such as T-SNE) do not.

Problem 22. Can PCA be used to reduce the dimensionality of a highly nonlinear dataset?

Jawab: PCA can be used to significantly reduce the dimensionality of most datasets, even if they are highly nonlinear, because it can at least get rid of useless dimensions. However, if there are no useless dimensions–as in a Swiss roll dataset–then reducing dimensionality with PCA will lose too much information. You want to unroll the Swiss roll, not squash it.

Problem 23. Suppose you perform PCA on a 1,000-dimensional dataset, setting the explained variance ratio to 95%. How many dimensions will the resulting dataset have?

Jawab: That's a trick question: it depends on the dataset. Let's look at two extreme examples. First, suppose the dataset is composed of points that are almost perfectly aligned. In this case, PCA can reduce the dataset down to just one dimension while still preserving 95% of the variance. Now imagine that the dataset is composed of perfectly random points, scattered all around the 1,000 dimensions. In this case roughly 950 dimensions are required to preserve 95% of the variance. So the answer is, it depends on the dataset, and it could be any number between 1 and 950. Plotting the explained variance as a function of the number of dimensions is one way to get a rough idea of the dataset's intrinsic dimensionality.

Problem 24. In what cases would you use vanilla PCA, Incremental PCA, Randomized PCA, or Kernel PCA?

Jawab: Regular PCA is the default, but it works only if the dataset fits in memory. Incremental PCA is useful for large datasets that don't fit in memory, but it is slower than regular PCA, so if the dataset fits in memory you should prefer regular PCA. Incremental PCA is also useful for online tasks, when you need to apply PCA on the fly, every time a new instance arrives. Randomized PCA is useful when you want to considerably reduce dimensionality and the dataset fits in memory; in this case, it is much faster than regular PCA. Finally, Kernel PCA is useful for nonlinear datasets.

Problem 25. How can you evaluate the performance of a dimensionality reduction algorithm on your dataset?

Jawab: Intuitively, a dimensionality reduction algorithm performs well if it eliminates a lot of dimensions from the dataset without losing too much information. One way to measure this is to apply the reverse transformation and measure the reconstruction error. However, not all dimensionality reduction algorithms provide a reverse transformation. Alternatively, if you are using dimensionality reduction as a preprocessing step before another Machine Learning algorithm (e.g., a Random Forest classifier), then you can simply measure the performance of that second algorithm; if dimensionality reduction did not lose too much information, then the algorithm should perform just as well as when using the original dataset.

Problem 26. Does it make any sense to chain two different dimensionality reduction algorithms?

Jawab: It can absolutely make sense to chain two different dimensionality reduction algorithms. A common example is using PCA to quickly get rid of a large number of useless dimensions, then applying another much slower dimensionality reduction algorithm, such as LLE. This two-step approach will likely yield the same performance as using LLE only, but in a fraction of the time.

Problem 27. How would you define clustering? Can you name a few clustering algorithms?

Jawab: In Machine Learning, clustering is the unsupervised task of grouping similar instances together. The notion of similarity depends on the task at hand: for example, in some cases two nearby instances will be considered similar, while in others similar instances may be far apart as long as they belong to the same densely packed group. Popular clustering algorithms include K-Means, DBSCAN, agglomerative clustering, BIRCH, Mean-Shift, affinity propagation, and spectral clustering.

Problem 28. What are some of the main applications of clustering algorithms?

Jawab: The main applications of clustering algorithms include data analysis, customer segmentation, recommender systems, search engines, image segmentation, semi-supervised learning, dimensionality reduction, anomaly detection, and novelty detection.

Problem 29. Describe two techniques to select the right number of clusters when using K-Means.

Jawab: The elbow rule is a simple technique to select the number of clusters when using K-Means: just plot the inertia (the mean squared distance from each instance to its nearest centroid) as a function of the number of clusters, and find the point in the curve where the inertia stops dropping fast (the ”elbow”). This is generally close to the optimal number of clusters. Another approach is to plot the silhouette score as a function of the number of clusters. There will often be a peak, and the optimal number of clusters is generally nearby. The silhouette score is the mean silhouette coefficient over all instances. This coefficient varies from +1 for instances that are well inside their cluster and far from other clusters, to -1 for instances that are very close to another cluster. You may also plot the silhouette diagrams and perform a more thorough analysis.

Problem 30. What is label propagation? Why would you implement it, and how?

Jawab: Labeling a dataset is costly and time-consuming. Therefore, it is common to have plenty of unlabeled instances, but few labeledinstances. Label propagation is a technique that consists in copying some (or all) of the labels from the labeled instances to similar unlabeled instances. This can greatly extend the number of labeled instances, and thereby allow a supervised algorithm to reach better performance (this is a form of semi-supervised learning). One approach is to use a clustering algorithm such as K-Means on all the instances, then for each cluster find the most common label or the label of the most representative instance (i.e., the one closest to the centroid) and propagate it to the unlabeled instances in the same cluster.

Problem 31. Can you name two clustering algorithms that can scale to large datasets? And two that look for regions of high density?

Jawab: K-Means and BIRCH scale well to large datasets. DBSCAN and Mean-Shift look for regions of high density.

Problem 32. Can you think of a use case where active learning would be useful? How would you implement it?

Jawab: Active learning is useful whenever you have plenty of unlabeled instances but labeling is costly. In this case (which is very common), rather than randomly selecting instances to label, it is often preferable to perform active learning, where human experts interact with the learning algorithm, providing labels for specific instances when the algorithm requests them. A common approach is uncertainty sampling (see the description in "Active Learning").

Problem 33. What is the difference between anomaly detection and novelty detection?

Jawab: Many people use the terms anomaly detection and novelty detection interchangeably, but they are not exactly the same. In anomaly detection, the algorithm is trained on a dataset that may contain outliers, and the goal is typically to identify these outliers (within the training set), as well as outliers among new instances. In novelty detection, the algorithm is trained on a dataset that is presumed to be "clean," and the objective is to detect novelties strictly among new instances. Some algorithms work best for anomaly detection (e.g., Isolation Forest), while others are better suited for novelty detection (e.g., one-class SVM).

——— "*Genius = 1% Inspiration + 99% Perspiration ...*" -Thomas A. Edison ———