Student Number: 181441 & Kaggle Team Name: jk446

1. Approach

This approach uses an RBF kernel SVM.

An SVM attempts to seperate data by drawing a hyperplane between the classes. It decides on this hyperplane based on two factors: maximising the margin (minimising the weight vector) between the hyperplane and the closest correctly classified samples and minimising the error cost of misclassified samples. These closest samples are called support vectors as the hyperplane is constructed based on these vectors. By tweaking a parameter , C, we can increase or decrease the importance of the two factors, or in other words, how regularised our model is. Picking a suitable C value is crucial to having the model generalise to unseen data [7].

Finally a "kernel trick", here the radial basis function, can be used to efficiently transform the data into higher dimensions where it may be more linearly separable. RBF uses a parameter γ which, in simplified terms, affects the distance at which sample can influence the decision boundary [2].

I chose SVM as it is "Effective in high dimensional spaces."[1] and to use a kernelised version as even with the high dimesionality, the data was not linearly seperable.

2. Methodology

2.1. Preprocessing

To make the model converge faster on a solution I used feature-wise standardisation $(mean = 0, var = 1)$.

I also used PCA to reduce complexity and noise by combining linearly correlated feactures. The number of components was chosen as the minimum possible without cross validation accuracy degradation (200).

2.2. Additional data

The problem with the additional train data is the large amount of missing values. To evaluate the effectiveness of various imputation methods: I took the fully complete dataset, randomly replaced values with NaN, ran each method on the now incomplete data and measured the mean squared error compared to the true values, see table 1. KNN

Iterative	Soft	KNN	Mean
0.1110	0.2213	0.1061	0.1510

Table 1. MSE of imputation methods 4.s.f.

peformed the best. Unfortunately KNN is quite time intensive so a single run is performed on the entire dataset, instead of calculating from only using k-fold train sets like the other transforms used. This leaking is data, however I feel this is the lesser of two evils compared to using a less accurate imputation method.

2.3. Annotation confidence and Class imbalance

A simple way of using the annotation confidence is to it treat as how sure we are that an image is sunny. If an image has a high confidence, it is more likely to have the correct label, therefore we should give a higher penalty for misclassifying it. We can do this by adjusting the regularisation parameter, C, on a per sample basis (sample weights). However, in Table 2 we can see that the confidence values are not evenly distributed, this means the classifier will bias towards certain classes or confidence values. We can fix

Class	Confidence	Frequency
0	0.66	465
0	1.00	570
1	0.66	1105
1	1.00	450

Table 2. Distribution of confidence annotations

this by undersampling until we have equal confidence proportions for each class . Figure 2 shows that after past a certain amount of data, accuracy stops increasing so there is no downside to undersampling. We can also fix the class imbalance (which is in the opposite direction to the test set imbalance) by undersampling too.

2.4. Test proportions

The test proportions have a heavy class imbalance, approximately $70 : 30$, and our classifier has a relatively low classifier accuracy against balanced data, $\sim 70\%$. Due to these factors I decided that a greater overall accuracy could be achieved by making the train data "look" like the test data by weighting classes by the test proportions, biasing the classifier to pick the majority class more often. This can be achieved, again by per-sample-regularisation. Having double the C value for a sample is mathematically equivalent to having 2 copies of that sample.

2.5. Domain adaptation

The domain adaptation problem can be seen as selection bias. Our training data was only selected from seaside locations rather than urban. Some of our training samples may be more similar to the urban test samples than others. We can importance weight our samples based on how similar they are to the test samples and multiply our sample weights by this importance, giving a higher cost to misclassifying more important samples. To calculate these importance weightings I used kernel mean matching[5].

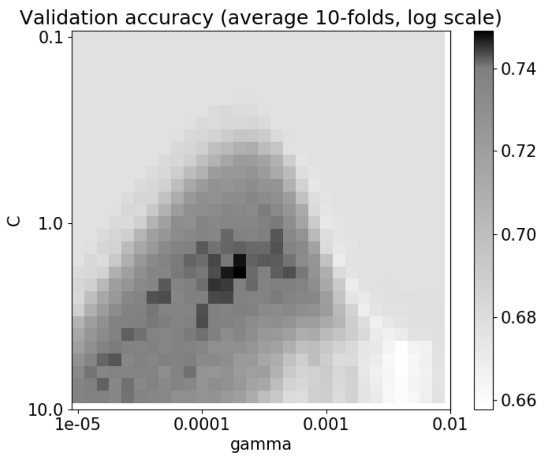

Figure 1. Heatmap of accuracy for parameter combinations, log scale. Best: $C = 2.043, \gamma = 0.0002212$ 4.s.f.

2.6. Model selection

For model selection I performed a grid search of C and γ. For each unique combination a stratified cross validation ($k = 10$) was performed, the average accuracy of these k-folds becoming the score of the combination. The accuracy scorer uses the domain importance and class proportions weightings (but not the confidence) as we want accuracy against a set that ”looks” like the test set. The combination with the highest score was used to retrain the classifier on the entire data set.

3. Results

3.1. Gridsearch

I performed a search of parameters around the sklearn suggested values, $C = 0.1, \gamma = n\_features^{-1}$[3].

We see in Figure 1 that the best combination is within the search range (not at the borders), so I don't feel the need to extend the ranges. However, it is possible that we are seeing a local maximum rather than a global one.

The Sklearn documentation suggests that for many C values there is a given gamma value that will give near optimal performance (i.e. $C \cdot \gamma = K$, where K is some constant) [2]. As a more in-depth search can be done with 1 parameter, I also ran a search of C while keeping gamma constant. This result is my second final submission.

3.2. Usefulness of additional resources

I found the additional training data to be very useful, you can see from Figure 2 that accuracy improves past the size of the size of the original training set.

The test label proportions is also very useful, by making my k-fold-test sets ”look” like the test set my cross validation scores were close ($\sim \pm 2\%$) to the kaggle submission scores meaning I could test parameters effectively.

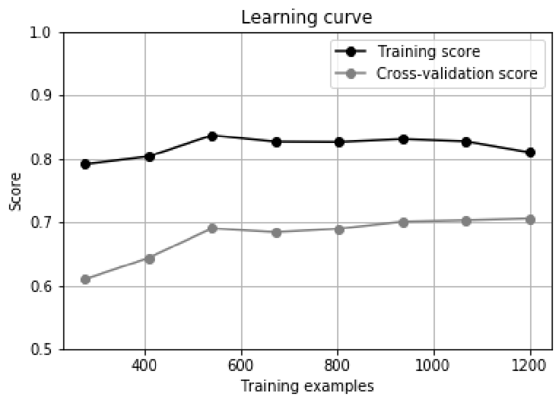

Figure 2. Accuracy (without sample-weights) against number of samples trained on.

Using the confidence annotations resulted in a modest but noticeable increase in accuracy.

Domain adaptation showed a very small improvement of accuracy, but small enough to be explained by noise.

3.3. Improvements

Iterative imputation could have made for a more accurate impute on the full dataset, unfortunately I couldn't test it due to how expensive it is to compute. I also could have gridsearched to possibly find a better k for the KNN impute.

Figure 2 shows a high variance, even after regularisation and dimensionality reduction. It has been shown that using an SVM bootstrap aggregation ensemble can outperform a single SVM by reducing variance [6]. The idea being you take many SVMs each trained on a smaller random subset of samples (with repeats), then take the aggregate of their predictions. Another ensemble method that could work is using different types of classifiers (svms with different kernels, random forests, logistic regression, etc.), training them on the whole data set and taking the majority vote.

If an ensemble method could take advantage of more of the training data, SMOTE[4] could be an effective way of fixing the class imbalance by generating extra artificial samples.

It would have been more accurate to represent a 0.66 class 1 sample as both a 0.66 class 1 sample and a 0.33 class 0 sample. Unfortunately there were practical difficulties in implementing this into my code. Also the confidence values have low precision with only 3 voters, I doubt there is a symmetric distribution around 66% confidence. A grid search could be used to find the best average mapping.

References

[1] 1.4 support vector machines; sklearn user guide. https://scikit-learn.org/stable/modules/svm.html. 1

[2] Rbf svm parameters; sklearn examples. https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html. 1, 2

[3] sklearn.svm.svc; sklearn documentation. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. 2

[4] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *JAIR*, 2002. 2

[5] J. Huang, A. J. Smola, A. Gretton, K. M. Borgwardt, and B. S. Scholkopf. Correcting sample selection bias by unlabeled data. *NIPS*, 2006. 1

[6] H.-C. Kim, S. Pang, H.-M. Je, D. Kima, and S. Y. Bang. Support vector machine ensemble with bagging. *SVM 2002*, 2002. 2

[7] A. Ng. Cs229 lecture notes; part v; support vector machines. http://cs229.stanford.edu/notes/cs229-notes3.pdf. 1