# Student Number: 178621 & Kaggle Team Name: Kristina Harper

## 1. Introduction

The task for this project was to utilize data to create predictions in a binary classification problem, which is submitted into a Kaggle competition. The competition submissions are ranked based on Classification Accuracy, and each prediction set is given a visible accuracy score based on 25% of the test data.

## 2. Approach

In order to generate solutions for this task, I chose to train a Support Vector Machine classifier implemented with the SciKitLearn toolbox in Python because of the high dimensionality of the data. The SVM utilizes the kernel trick, which implicitly maps inputs into a high-dimensional feature space. In addition, the data, and particularly the CNN features, appeared sparse and contained many zero values. Because the SVM identifies support vectors and uses only those to optimize the decision boundary, it is a logical choice when implemented with a soft margin and loss function to punish misclassified data points [4].

Another reason for choosing an SVM classifier was due to the binary predictions of the data; using a maximum-margin classifier to create a hyperplane separating two classes of data seemed intuitive. SVM implementation also offered useful weighting features that aided in incorporating the additional data provided. These features will be explained in detail in the Methodology section. The SVM's robustness and accuracy given sparse, high dimensional data made it a logical choice to implement for this task.

An SVM works by plotting each data item as a point in n-dimensional space, where n represents the number of features of the data. The classification occurs by finding a hyperplane that acts as a decision margin between the two classes. The hyperplane maximizes the distances between the nearest data points and the hyperplane, which corresponds to minimizing the weight vector [4]. Support vectors refer to the coordinates of data points along the decision margin or those that are misclassified [6]. The decision boundary is defined as a linear combination of the support vectors [4]. The general equation to maximize the SVM margin is below.

$$\begin{aligned} &minimize_{w \epsilon R^d} \parallel W \parallel \\ subject-to-for-n &= 1, ..., N \\ &y^n \langle w, x^n \rangle \geq 1 \end{aligned} \tag{1}$$

Support Vector Machines can utilize the kernel trick to create the decision hyperplane and implicitly create a high dimensional mapping. The Kernel trick is a function which takes a low dimensional input space and transforms it into a higher dimensional space, allowing the problem to become separable where it wasn't before. This trick relies on the assumption that taking a dot product and exponentiating gives the same results as mapping directly into the high dimensional space and taking the dot product [3]. This method aids in memory efficiency, increases the speed of computation, and allows flexibility in the kernels, as the kernels represent different measures of similarity, some of which may be more appropriate for the data than others.

$$max_{\lambda_1,...\lambda_n} \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n,k=1}^{N} \lambda_n \lambda_k y_n y_k k(x_n, x_k) \tag{2}$$

$$...\text{subject to } \lambda_n \geq 0, \forall n = 1, ..., N$$

In other words, to transform our existing data to higher dimensional data, which can produce more accurate classification, we don't need to compute the exact transformation of our data, which would be very computationally expensive [8]. Instead, we can compute the inner product of our data in the higher dimensional space [7]. We do this by minimizing the weight vector as a function of lambda. Given *Equation 2*, the parameters representing the penalty parameter of the error term (C in Python) and the coefficient for the non-linear decision boundary in SVM kernels - $\lambda$ in the equation and $\gamma$ in the python SVM implementation - must be optimized [4].

Thus, Support Vector Machines can deal with data of high dimensionality through the use of this kernel trick.

## 3. Methodology

I first implemented an SVM with a linear kernel. I ran the SVM only on the training data with no pre-processing to get baseline results, which I obtained by cross-validation; I tested these first results on the validation set. This simple, linear classifier generated predictions with 0.75 accuracy. For this model and all those mentioned later, test predictions were uploaded to the Kaggle competition. Given that accuracy percentages shown on Kaggle only represent the model's performance on a small subset of the test data, analysis will focus on model performance on validation sets. This will presumably be a more accurate prediction because all of the validation data will be visible for analysis.

Since there are many parameter settings to optimize within a Support Vector Classifier, before attempting trial-and-error to manually explore parameters, which would be

time-consuming and tedious, I used the Grid Search function to do model selection using five folds via the cv=n parameter. I first tested parameters 'C': 1,10,100,1000, kernel: 'linear';'rbf', and $\gamma$: 0.001,0.0001. I then applied the optimal model output - 'rbf', 'C'=10, and $\gamma$=0.0001 - on the unprocessed training data set and tested on the validation set. The validation score output was 0.807 *(Table 1)*.

### 3.1. Pre-processing

Preprocessing was the next step towards improving classifier accuracy. I processed the training data using standard scaling, which standardizes features by removing the mean and scaling to unit variance. Centering and scaling occur independently on each feature when using the StandardScaler function in SKLearn. This is beneficial to the Support Vector Machine because the learning algorithms within the RBF kernel assume that all features are centered around zero and have variance in the same order. Standardizing variance prevents the dominance of one feature with high variance magnitude within the objective function.

After standardizing the data, I implemented dimensionality reduction. Although a little information is lost through this process, transforming the data and projecting it into a lower dimensional space made sense because of the high number of features in the dataset. To lose as little information as possible, I input the scaled data into a Principal Component Analysis (PCA) [2], [1]. Through this new projection, we can access the components that contain the most variance and are therefore likely to be important in classifying the data points. Initially, I accounted for 95% of the data's variance when calculating the PCA.

My results in Model 4 using the LinearSVC function of SKLearn, both with regard to the validation set results and visible test set results from Kaggle, were significantly worse than in Model 3. As a result, in the next steps of my methodology, I kept the svm.SVC() classifier as an option but no longer included LinearSVC as a testing parameter in my grid search. I attempted to keep the trial parameters to a minimum, because each run of the grid search algorithm to optimize hyper-parameters was time-consuming.

### 3.2. Utilize Additional Data

After conducting trials on and applying pre-processing techniques to the training data, I began integrating the additional data into the classifier to evaluate the results on the validation set. Because there are only 456 labeled training instances in the training data set and the test data contains 5040 instances, including more labeled data to train the model would presumably decrease the sparsity problem and improve classifier accuracy. The challenge in integrating this data is that there were missing values for the features. After appending the additional data to the training data set dataframe in my workspace, I conducted imputation and filled the missing not-a-number (NaN) values with the mean of each column. I then split the 4560 data points into training and validation sets and ran a grid search on the new full data set. The grid search included a Pipeline in which scaling, PCA, and classification were conducted. The search space included variables for the number of PCA components (1,2,3,10,100,200,500), C value for the classifier (1,10,100), gamma ($\gamma$) value for the classifier (0.0001, 0.001, 0.01). Class weight was also included in the parameter list, but this was fixed at {0:0.5714, 1:0.4286}. The class weight parameter within the grid search pipeline represents the penalty for each misclassified data point. I set these at the values given in the test proportions file, which represents the number of positive (snowy) and not-positive data points in the test set. This proportion is reversed in the training dataset, so I increased the weight on non-positive data points to better represent the testing data.

The best model returned after the grid search completed was and RBF kernel with parameters C=1, gamma=0.0001, and PCA n-components=200. These parameters were implemented in Model 7, with the additional parameter of sample weighting during model fitting. Sample weighting was set to the confidence values provided in the annotation-confidence file. This means that when the algorithm was fitting given the parameters above, a stronger weight was given to those data points with higher confidence values (1) versus those with lower confidence (0.66) and the classifier will put more emphasis on classifying high-confidence points correctly. Implementing these parameters and weighting values provided the best results thus far *(Table 1)*.

### 3.3. Full Exploration of Hyper-parameters

In order to fully explore hyper-parameters, I ran a grid search using a higher number of PCA components and lower gamma values. The optimal model output is represented by Model 8 and recommended C=10, gamma = 0.00001, and 2000 components with PCA.

I had received fairly good prediction results with an RBF kernel in my SVM classifier given the validation score output, but given that I had only thoroughly considered an RBF kernel and associated parameters via grid search, I ran a grid search of various kernels, specifically linear, poly, and sigmoid. Using 5-fold cross-validation, I kept other parameters narrow relative to those that had provided most accurate testing results in prior trials (n-Components: 1,2,3,100,200; C:1,10; gamma: 0.0001,0.001). After testing these parameters, I implemented the best one, Model 9 *(Table 1)*.

| SVM Model | Preprocessing | Score |
|---|---|---|
| (1)Linear | n/a | 0.754 |
| (2)RBF (C=10, $\gamma$=0.0001) | n/a | 0.807 |
| (3)RBF (C=10, $\gamma$=0.0001) | S, PCA(0.95) | 0.754 |
| (4)LinSVC (l2,hinge, c=10) | S, PCA(0.95) | 0.737 |
| (5)RBF all(c=10,$\gamma$=0.0001) | n/a | 0.750 |
| (6)RBF all(c=10,$\gamma$=0.0001) | S, PCA(0.95) | 0.800 |
| (7)RBF all(c=1,$\gamma$=0.0001) | S, PCA(200) | 0.832 |
| (7.1)RBF all(c=1,$\gamma$=0.0001) | S, PCA(200) | 0.823 |
| (7.2)RBF all(c=1,$\gamma$=0.0001) | S, PCA(200) | 0.833 |
| (8)RBF all(c=10,$\gamma$=0.00001) | S, PCA(2000) | 0.826 |
| (9)Lin all(c=10,$\gamma$=0.0001) | S, PCA(200) | 0.825 |
| (10)RBF train(c=1,$\gamma$=0.0001) | S, PCA(200) | 0.772 |
| (11)Model 7,no ClassWeight | S, PCA(200) | 0.823 |
| (12)Model 7,no SampleWeight | S, PCA(200) | 0.815 |

Table 1. Representation of validation scores for models implemented with different parameters. 'S' in the preprocessing column stands for Standard Scaling. The value in parenthesis next to PCA represents the number of components. Models with 'all' specified utilize all additional data in training. Model 7 gave the most accurate validation predictions and was reproduced in Models 7.1 and 7.2 with re-implemented scalers. The numbers in the table do not correspond to test prediction file numbers uploaded to Kaggle.

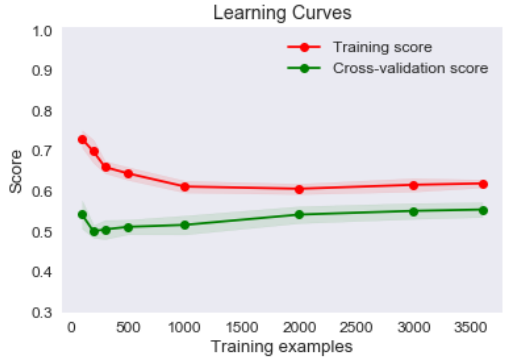


Figure 1. The learning curve for the best model: SVM classifier with RBF kernel, C=1, $\gamma$=0.0001. Preprocessing was Standard Scaling and PCA with 200 components. The graph shows that the model has high bias and low variance.

## 4. Results and Discussion

### 4.1. Preprocessing Parameters

It is clear that the preprocessing methods of Scaling and PCA significantly improved the classification power of the model for implementations with additional-training data. A comparison of Model 5 to Model 6, where all conditions are consistent except the implementation of pre-processing, demonstrates this performance improvement *(Table 1)*. Interestingly, optimizing the number of components of PCA at 200 also improved the performance of the algorithm.

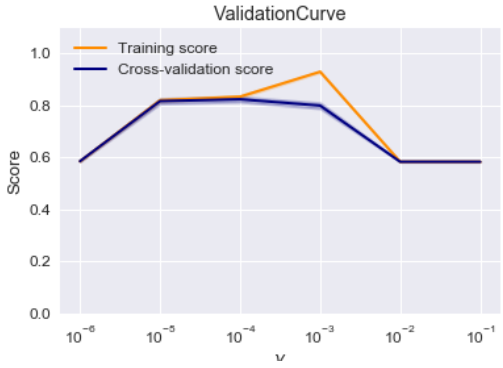


Figure 2. The validation curve for gamma ($\gamma$) in the best predictive model: Model 7. The curve shows that varying the gamma parameter results in over-fitting where the training score is high and validation score is low, at $10^{-3}$.

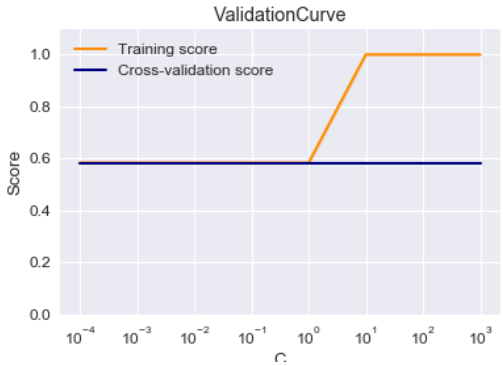


Figure 3. The validation curve for the C parameter in the best predictive model: Model 7. The curve shows that varying the C parameter results in over-fitting where the training score is high and validation score is low, at $10^{1}$ and higher.

When 95% variance is accounted for in the Principal Component Analysis on all available scaled data, 1680 components are used to train the SVM. Including 200 components for all data accounts for approximately 68% of variance for the best model produced according to cross-validation scores: Model 7.2. It is interesting, and somewhat counterintuitive, that including data that accounts for less variance actually improves the predictive power of the algorithm. This could be because including PCA with fewer components filters out noise and redundancy while focusing on the combinations of features that are most important for classification.

### 4.2. Hyper-Parameters & Weighting

Hyper-parameter C was optimized at 1 when all data was used to fit the model. This penalty parameter, however, was manipulated with the class-weight parameter in

the SVM. Class-weight values were consistent for Models 7-10 and set at {0:0.5714, 1:0.4286}, as mentioned above. This weighting feature is explained in greater detail in the Methodology section. When all other hyper-parameters are kept consistent and no class weight used, the score is 0.823 (Model 11), which is not as accurate as when class weight is implemented. Removing sample weighting alone from the fit method of the best performing model gives a cross-validation score of 0.815, shown in Model 12. To determine if the model was over-fitting with regards to the C parameter, I created a validation curve. The figure shows that varying the C parameter results in over-fitting at values of 10 and greater, and that the optimal parameter is likely to be 1 with all other parameters of the best model consistent *(Figure 3)*.

The gamma ($\gamma$) parameter was optimized at 0.0001 ($10^{-4}$) with an RBF kernel and C at 1 for the best model, and I explored the possibility that the model was over-fitting with regards to this parameter *(Figure 2)*.The gamma validation curve shows that the model does learn the data well for gamma of $10^{-5}$ and $10^{-4}$, but begins to over-fit the training data when gamma is $10^{-3}$. This confirms the grid search best-model output and the cross-validation scores shown in Table 1.

### 4.3. Bias-Variance Trade-Off

In order to diagnose where the best model stood regarding the bias-variance trade-off, I generated a learning curve to plot an increasing number of training examples against the score [8]. If a model has low bias, it can capture differences in data, and if it has high bias, the model is too simple and does not fit the data well. This corresponds to low variance, in which the model does not change much as training sets change. Low bias usually means that the model has higher variance, when the model is too complex, and small changes to the data result in major changes to the solution or predictions. Ideally, a model would have low bias and low variance, but this is impossible. Low bias is desirable because the model will be complex enough to represent the data well, but this may not be generalizable and result in over-fitting to the training data [5]. Low variance would temper the complexity of a model to better filter out noise and avoid over-fitting.

The learning curve for the best model shows that the model has low variance and high bias. It does not show over-fitting to the training data, but may be too general, as there is clearly not a large increase in model accuracy as more samples are added. If the model were over-fitting, the graph would show that the training curve in red approaches a score of 1.0 *(Figure 1)*.

Although the learning curve shows that adding additional training data does not greatly improve the predictive power of the model or show that the model is over-fitting, a comparison of the cross-validation results shows the importance of adding the additional training data to model training. Model 10 implements the same parameters and preprocessing techniques as the best performer, Model 7. But, whereas Model 7 uses all of the training data, including the additional data with mean-filled features, Model 10 uses only the data from the training file. The cross-validation accuracy of Model 10 is 0.772, significantly poorer than the best performing model.

## 5. Future Work

In the future, I would test different methods of dealing with the missing variables within the additional-training data set, such as with median and mode. A more precise option to the imputation I conducted with mean values is Similar Case Imputation, in which the mean for data instances with prediction '0' and for prediction '1' is used to impute missing features. Because instances with the same label are more likely to have similar feature values than instances with the opposite label, this method might provide more accurate substitutions. Another option is to build a model to predict the missing values in the data set, through KNN for example. In this method, missing values of an attribute are imputed using a number of attributes most similar, given a distance function, to the attribute whose values are missing. I would like to detect and treat data outliers because they can skew the data and reduce the classifier's accuracy. I would also like to more deeply explore the difference between CNN and GIST features and determine which set of features has greater predictive power. A more thorough exploration of the data through these methods could provide better predictions than those that the models above generate, and would result in a more comprehensively-tested model.

This project was the first in Machine Learning in which I used Python; my prior work in Python was limited to Natural Language Processing tasks. Implementing the techniques outlined above and exploring the tools available within SKLearn allowed me to better understand the mechanisms behind Support Vector Machines and kernels. In addition, applying grid-search and a pipeline to optimize parameters was an energizing and exciting experience, as in the past I had explored parameters through trial-and-error. I appreciated that this task allowed me to manipulate data for classification using any and all tools available; the freedom and competition aspects motivated me to get the best results possible.

The variation in classification results displayed by the Kaggle leader board versus my own validation score prompted me to explore over-fitting and question which model really displayed the best predictive power. I did achieve my goal of achieving over 80% accuracy in cross-validation, but the greater feeling of achievement came through actually implementing a Machine Learning model and experimenting with the ways it could be optimized.

## References


[1] E. Alpaydin. *Introduction to machine learning*. MIT press, 2014. 2

[2] L. Berthouze. Machine learning: Lecture week 2 - pre-processing, 2018. 2

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992. 1

[4] P. Flach. *Machine Learning: The art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012. 1

[5] N. Quadrianto. Machine learning: Lecture week 8 - regression, 12 April 2018. 4

[6] N. Quadrianto. Machine learning: Lecture week 6 - support vector machine, 15 March 2018. 1

[7] N. Quadrianto. Machine learning: Lecture week 7 - kernel methods, 22 March 2018. 1

[8] S. Shalev-Shwartz. *Understanding machine learning : from theory to algorithms*. 2014. 1, 4