Classifying Forest Cover Types - Comparing Support Vector Machines with Multilayer Perceptron Models

1. Description and Motivation of the Problem

This study aims to compare and contrast the performance of Support Vector Machines (SVM) with Multilayer Perceptron (MLP) models at predicting different forest cover types. The prediction is a multi-class classification problem, based on environmental data representing seven imbalanced forest cover types found in the Roosevelt National Forest.

Forest cover type prediction is important for land management agencies as it reflects the ecological balance and diversity of an area. Alternatives to computer-based modelling involve direct recording and estimates based on remote sensing data – however both approaches have limitations in terms of cost, time and prediction accuracy[1]. It may also be necessary to predict cover distribution in areas that are not legally accessible, hence the need for accurate models.

I will compare my results to the current equivalent standards found in the literature; Blackard et al.[1], who utilized an artificial neural network (ANN), and Trebar and Steele[2], who applied distributed SVM architectures to multiple binary classifications.

2. Description of the Dataset

The dataset is the Covertype Data Set from the UCI Machine Learning Repository[3]. The data has been compiled from a combination of US Geological Survey and US Forest Service Data and includes 581,012 observations categorized across 7 different forest cover types.

The distribution of the target classes in the dataset can be seen in Figure 1, with class 1 (Spruce/Fir) and 2 (Lodgepole Pine) comprising 85% of the total observations. The largest imbalance is seen in class 4 (Cottonwood/ Willow) which makes up 0.47% of the overall data. Each class represents a different tree type found in the surveyed ~160,000-acre area, excluding class 7 (Krummholz) which is an amalgamation of 3 separate species.

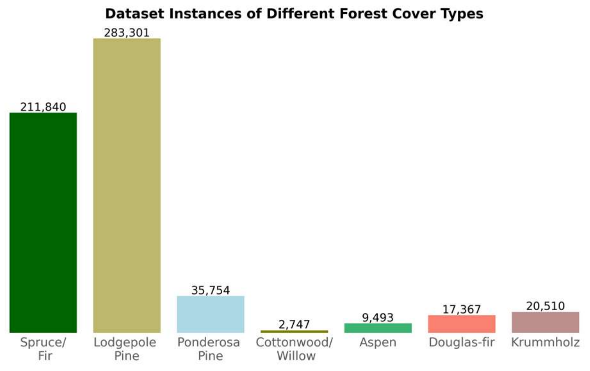


Figure 1: Bar chart showing distribution of datapoints amongst the seven target classes

There are 54 predictor variables. 44 of these are binary variables; 40 of which refer to the presence of specific soil types, while the remaining 4 relate to defined wilderness areas within the forest. The remaining 10 predictors are continuous quantitative variables, detailed in Figure 2 below:

Variable Description	Mean	Std. Dev.	Min.	Max.
Elevation (m)	2,959.4	280.0	1859	3,858
Aspect (azimuth from true North)	156.7	111.9	0	360
Slope (°)	14.1	7.5	0	66
Horizontal distance to nearest surface water feature (m)	269.4	212.5	0	1397
Vertical distance to nearest surface water feature (m)	46.4	58.3	-173	601
Horizontal distance to nearest roadway (m)	2,350.1	1,559.3	0	7117
Relative measure of sunlight at 9am (0-255 index)	212.1	26.8	0	254
Relative measure of sunlight at Noon (0-255 index)	223.3	19.8	0	254
Relative measure of sunlight at 3pm (0-255 index)	143.5	38.3	0	254
Horizontal distance to nearest historic wildfire ignition (m)	1,980.3	1,324.2	0	7,173

Figure 2: Summary of key statistics of continuous and discrete quantitative variables.

Correlation matrixes revealed a small number of correlated variables in the dataset. The collinearity of these features was investigated in more detail (see Figure 3) and I decided not to remove any from the data as the plots revealed non-linear correlations – especially for the Hillside, i.e., Relative sunlight, variables - between the individual classes that I felt would be valuable model inputs. Both approaches in the literature also kept all 54 features.

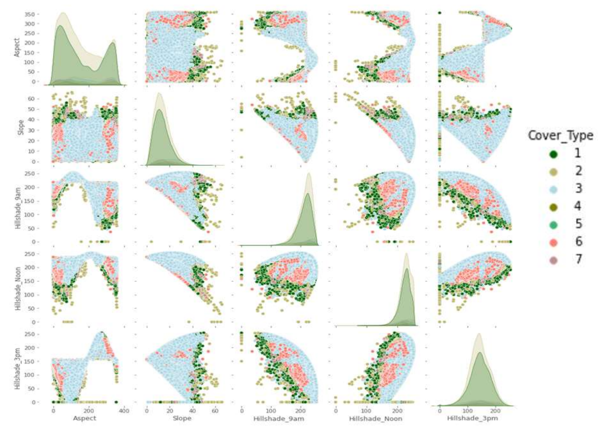


Figure 3: Pairplot of highly correlated (>0.5 magnitude) variables

3. i) Support Vector Machines (SVM)

SVMs create a line (in 2D space), or a hyperplane (3D+), that separates data into different classes. The model tries to maximize the margin, which is defined as the distance between the hyperplane and the closest datapoints (the support vectors) on either side, ensuring the greatest chance that new data is classified correctly.

When data is not linearly separable, SVMs utilize the kernel trick to project the data into a higher dimensional space where separation becomes possible. Each different kernel results in a uniquely-shaped decision boundary, influencing how datapoints are separated.

In order the apply SVMs to multi-class predictions there are two predominant approaches that combine several binary SVM classifiers – 'One-vs-one' and 'one-vs-all. The former approach was chosen for this paper and trains n(n-1)/2 SVM classifiers (where n is the number of classes), each designed to binarily predict one class against one other class. The final prediction is based on a max-wins voting system. Alternatively, the 'one-vs-all' approach trains n classifiers to identify each class versus a combination of all the other classes, relying on a winner-takes-all strategy to assign predictions.

Advantages

- Optimization through quadratic programming ensures one unique, global solution and avoids the problem of local minima.
- The trained model depends only on the support vectors which can increase efficiency of prediction as it only uses a subset of the data.
- Due to the sparseness of its solution, SVMs are less prone to overfitting compared to neural networks which rely on the full data.
- SVMs can be very effective for smaller datasets with high dimensionality. Their accuracy can outperform neural networks given the right dataset conditions[4].

Disadvantages

- However, quadratic optimization means the computing time can increase quadratically with number of examples[5].
- As the number of examples increase the kernel matrix can become very large, increasing memory requirements or requiring caching[5].
- For both these reasons, SVM can be extremely slow for large datasets.
- Where many support vectors and dimensions are required to separate the data, the model becomes difficult to interpret.
- It can be difficult to find a balance between the parameters – kernel type, gamma and C – and so search cost/ time can be high.

3. ii) Multilayer Perceptron

MLPs are feedforward neural networks that use backpropagation to update weights to train the network[6]. An MLP consists of an input and output layer, connected by a number of hidden layers. The input layer usually corresponds to the number of variables present in the dataset, whilst the output layer may have either a single output neuron (in the case of regression tasks or binary classification) or multiple output neurons equal to the number of target classes for multi-class classification.

Each layer of an MLP is fully connected and consists of a number of specified neurons. These neurons are linked and updated via a nonlinear activation function that is applied to the sum of the input signals for each neuron, as well as the synaptic weights associated with each neuron[7]. These output signals then propagate forward throughout the network. The neuron weights are initialized randomly and updated as error signals are back-propagated through the network, changing the weights towards the steepest local gradient[6]. After weights have been altered, the function signal propagates forward again and this cyclical process continues until the model has been trained.

Advantages

- Due to the non-linear activation function, MLPs with just one layer can approximate any function between the input and output layers[8].
- By utilizing batches, training time scales linearly with dataset size, offering speed advantages over SVM in instances of large datasets.
- High adaptability – the hidden layers and neurons can handle high complexity and heteroskedasticity in the data as well as being easily applied to multi-class classification.

Disadvantages

- Gradient descent and random weight initialization can result in optimization towards local minima rather than the global optimum.
- Neural networks are considered "black box" models[9], and so are unsuitable where model interpretability is highly desired.
- The fully connected nature of MLPs can result in redundant or inefficient connections between neurons needlessly slowing down the model.
- Where the model has more capacity than necessary it can overfit, leading to poor generalizability on test data. However, this can be offset by using early stopping criteria[9].

4. Hypothesis Statement

I expect that SVM with outperform MLP for this classification task due to the large imbalance present across the classes. The small number of samples in the minority classes may lead the MLP to overfit that data and generalize worse to unseen data, compared to an SVM.

The literature also suggests that SVM will outperform as Trebar and Steele[2] achieved accuracies of over 90% across their binary classifiers, whilst Blackard and Dean[1] achieved 70.58% blended accuracy with a single layer ANN. I expect my MLP to outperform the literature's ANN as the additional hidden layers should be able to model more complex relationships among the data.

To extend the comparison I will investigate the impact that different sampling techniques have on overall performance. A combination of over and under sampling can have a similar impact as cost-sensitive learning, and can result in greater sensitivity towards the classification of the minority classes[10].

I hypothesize that over sampling will have a greater positive effect on the performance of the MLP as it will provide additional minority samples to aid model training and potentially reduce the likelihood of overfitting. I expect sampling techniques to have less of an effect on the SVM because if the samples are/were not used as support vectors, adding/removing them will not impact the model's decision boundary.

5. Choice of Training and Evaluation Methodology

As the data is not ordinal, it will be split randomly, with 80% being used for the training data and 20% held back for testing. For the MLP, a further 20% of the training data will be separated for validation within epochs. A stratified sampling method will be utilized, to retain the same imbalance of classes across the subsets of data.

For both models the training data will be scaled between zero and one. The scaler model will subsequently be applied to the validation and test sets. Scaling prevents "attributes in greater numeric ranges dominating those in smaller numeric ranges"[11].

Four-fold cross validation performed on 10% stratified subsets of the training data was chosen to reduce overall search time. Four was chosen as the value of k to ensure the number of minority class samples in each fold wasn't too small for the model to learn.

Due to the different ranges of hyperparameters combinations I have chosen to search through for each model, I will employ different search methods. For the MLP it is feasible to use grid search to test different combinations of the 7 parameters chosen (see 6.ii)). However, for SVM I will utilize random search, which provides an efficient alternative to grid search, and allows me to test an exponential range of the 'C' parameter more effectively across different kernel types.

The evaluation criteria for training and hyperparameter searches will be cross-entropy loss for the MLP and cross-validation fold accuracy for the SVM, as SVM does not output class probability distributions. Early stopping criteria was added to the MLP whereby the model stopped training when there were 5 subsequent increases in validation loss, this helps prevent the model overfitting and reduces training time[9].

Overall accuracy has been chosen as the test set evaluation metric, in line with the reference papers, as no one forest cover type is more important than another to predict correctly. However, un-weighted (due to the class imbalance) F1 score, precision, recall metrics, as well as confusion matrices will be also used to compare overall classifier performance and the impacts of sampling methods.

6. i) Parameters and Experimental Results - SVM

The kernels chosen to search through were; polynomial, RBF, sigmoidal, and two implementations of the linear kernel – using the liblinear, designed for large-scale linear classification, and libsvm libraries. The RBF kernel was found to achieve the highest overall and average accuracy across the parameter searches. The RBF only marginally outperformed the polynomial kernel (by 1% on average), but was ~76% faster to train. The liblinear library was the fastest kernel, however the average accuracy was 12% worse than RBF – a trade-off not worth accepting. Comparisons can be seen in Figure 4.

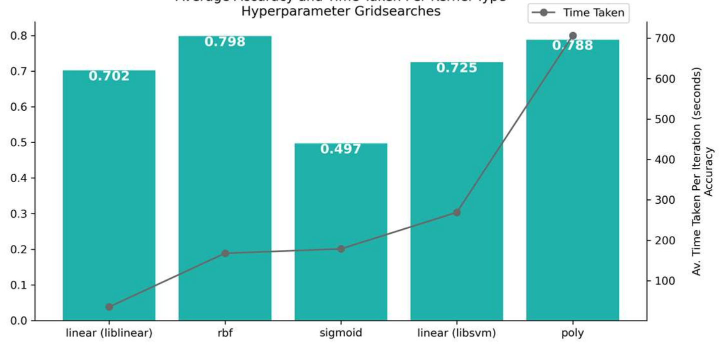


Figure 4:
Bar chart showing average SVM accuracy and line chart showing average time taken per set of hyperparameters tested for each kernel type.

The other parameters compared for SVM were the regularization parameter 'C' and the kernel coefficient 'Gamma'. Using the sklearn "RandomizedSearchCV" package, C was searched in an exponential distribution between 0 and 1000, while gamma was searched in a uniform distribution between 0.1 and 1. The distribution of model accuracies across the parameter values can be seen in Figure 5.

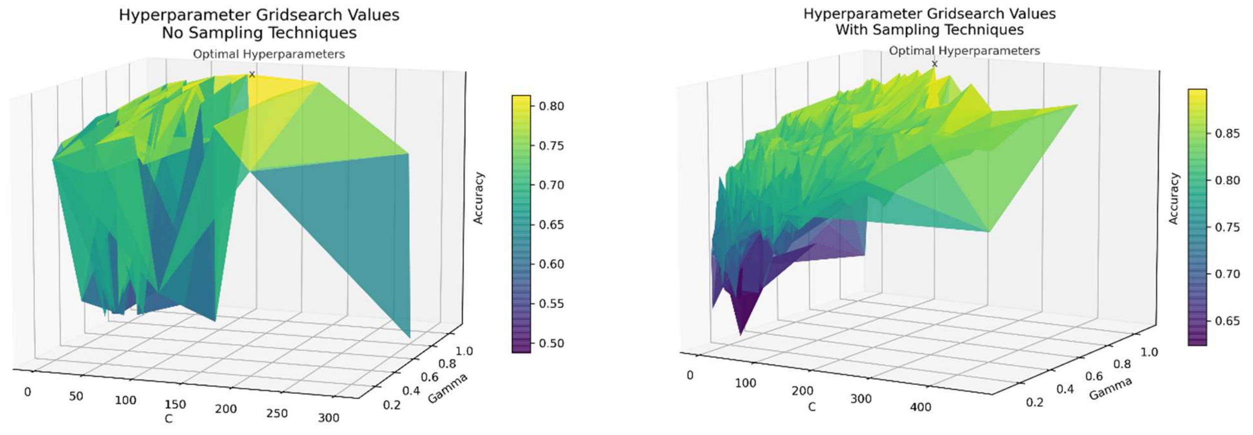


Figure 5: Surface plots showing SVM model accuracy across searched hyperparameters C and Gamma

6. ii) Parameters and Experimental Results - MLP

The hyperparameters chosen to vary for the MLP were; *learning rate, maximum number of hidden neurons in the largest layer, dropout, momentum/weight decay, number of layers, activation function and the optimization algorithm*.

Two and three-layer architectures were tested, however the performance of both was similar, with a slower computation time for the additional layer meaning two was selected for this problem.

The optimization algorithm is used to find parameter values that minimize the loss function (cross-entropy). The two optimizers I chose to try were Stochastic Gradient Descent (SGD) and Adam due to their popularity and effectiveness on a wide variety of problems[12]. Adam naturally incorporates a momentum element, so weight decay was tested as a parameter instead. Rectified Linear Unit (ReLU) and Leaky ReLU were tested as the activation functions within the hidden layers of the neural network.

The following ranges of parameters were searched through using a manual grid search for each combination of optimizer and activation function, with optimal parameters highlighted in Figure 6.

Learning Rate	***Maximum Hidden Neurons***	***Dropout***	***Momentum***
[0.1, $1e^{-2}$, $1e^{-3}$, $1e^{-4}$, $1e^{-5}$]	[32, 64, 128, 256, 512]	[0, 0.1, 0.2, 0.3]	[0, 0.1, 0.9, 0.99, 0.999]

Optimizer	**Activation Function**	**Minimum Cross-Entropy Loss**	**Parameters Associated with Min. Loss**			
			Learning Rate	*Momentum*	*Max Neurons*	*Dropout*
SGD	*ReLU*	0.203	0.1	0	256	0
	Leaky ReLU	**0.189**	**0.1**	**0**	**512**	**0.1**
Adam	*ReLU*	0.202	0.001	0 *(weight decay)*	512	0
	Leaky ReLU	0.201	0.1	0 *(weight decay)*	512	0.1

Figure 6: Hyperparameters required for the lowest cross-entropy loss for each optimizer-activation combination

6. iii) Sampling Techniques

Under sampling was performed by removing samples at random from the two majority classes (Classes 1 and 2) according to 6 different strategies, ranging from 15% removal to 80% removal. Tomek Links under sampling was also tested to remove "borderline" instances from the majority class that the algorithm considers likely to be misclassified. For oversampling, 7 different strategies were applied to the minority classes, ranging from a 20% increase, to equalizing the minority class samples with the largest majority class. Both synthetic Minority Over-sampling (SMOTE)[10] and Borderline SMOTE techniques were tested for their effectiveness.

Hyperparameter testing found that a combination of Tomek Links under sampling, 15% under sampling of the two majority classes, and SMOTE over sampling to equalize the minority classes, resulted in the best performance for both models. For SVM, maximum accuracy on the training subset was 13% higher (see Figure 5) after sampling techniques, whilst for MLP minimum cross-entropy loss was 34% lower. These optimal sampling parameters were applied to the test set in Section 7.

7. Analysis and Critical Evaluation of Results

Classifier	Sampling Techniques Used?	Overall Accuracy	Un-Weighted F1 Score	Un-Weighted Precision	Un-Weighted Recall
SVM – Baseline	No	0.773	0.59	0.75	0.55
SVM	No	0.865*	0.83	0.87	0.80
SVM	Yes	0.853	0.82	0.76	0.91
MLP – Baseline	No	0.725	0.4	0.41	0.42
MLP	No	0.883	0.83	0.82	0.83
MLP	Yes	0.935*	0.91	0.88	0.95

Figure 7: Test Set Performance- Shown by evaluation metrics per classifier, with and without sampling techniques

*Denotes the best model performance, as measured by overall accuracy, the associated model hyperparameters with these scores was as follows:

- **SVM:** Kernel = RBF, C = 108, Gamma = 0.988, no sampling techniques.
- **MLP:** Optimizer = SGD, Activation Function = Leaky ReLU, learning rate = 0.1, momentum = 0, dropout = 0.1, maximum hidden neurons = 512, with sampling techniques.

7. i) Performance:

Counter to my predictions, MLP outperformed SVM – the best MLP model scoring a 8% higher overall accuracy than the best SVM model.

The optimal MLP model is much better at predicting the minority classes, with an average accuracy of 95.33% across classes 3-7, compared to 77.15% for the optimal SVM model (see Figure 8). It is clearly benefitting more from the sampling techniques employed (described below in more detail).

MLP showed greater performance gain from altering hyperparameters compared to the baseline model, likely due to the advanced range of configuration options available. However, this also shows that SVM performs better ‘out of the box’, and requires less manipulation to achieve optimal results.

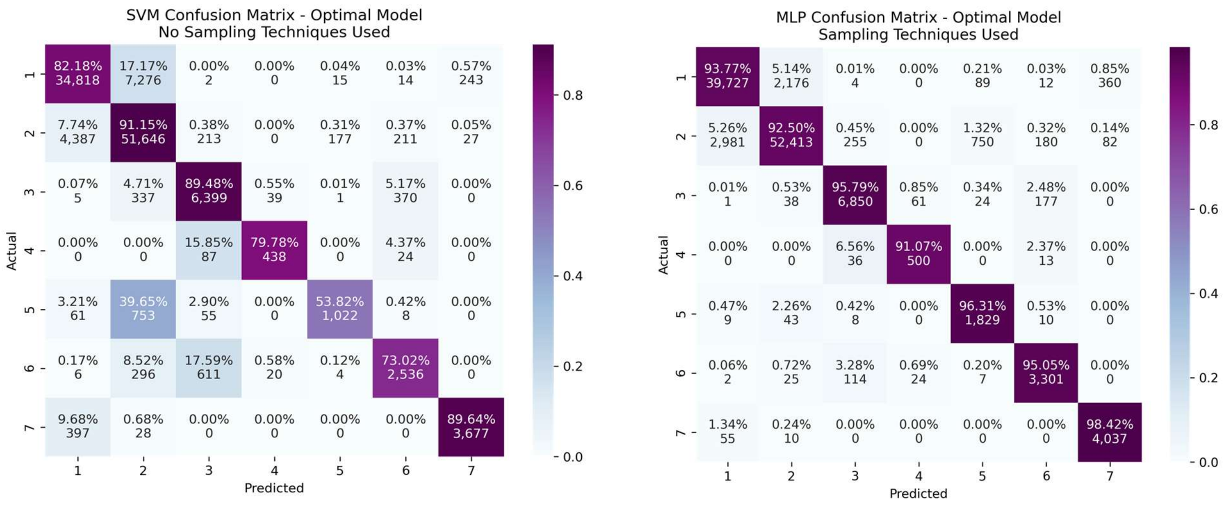


Figure 8: Confusion matrix plots for the best performing SVM and MLP models - percentage and counts included per class.

7. ii) Sampling Impact:

Sampling techniques did improve the recall of the SVM classifier, to the detriment of the other tracked metrics (see Figure 7). Better prediction of minority classes after applying sampling techniques (93.13%), was not beneficial enough to outweigh the worse prediction of the majority classes – 84.23% for the two majority classes, versus 86.67% in the optimal, non-sampling techniques model (Figure 8).

In this instance, the under-sampling techniques may have removed important majority class datapoints that were being relied upon as key support vectors when sampling techniques were not used.

Conversely, SMOTE oversampling, combined with Tomek Links and 15% under-sampling, were beneficial for the MLP model – improving overall accuracy, f1 score and recall, with only slightly worse precision. These techniques increased the overall training dataset size to 1.05mn from 317k.

This result aligns with my hypothesis that the additional training data added through SMOTE oversampling would be most beneficial for the MLP, which often performs better with larger datasets as more effective connections can be iteratively formed.

This likely led to the much-improved performance of predicting the minority classes as, in the most extreme example, the model was fed 315k additional samples compared to the original 2.7k in class 4. There was potentially less drawback from under-sampling the majority class for the MLP, as the remaining samples still provided enough data for the network to train effectively.

7. iii) Speed to Train and Test Final Models:

As Figure 9 shows, SVM became increasingly slow compared to MLP as the number of samples increased, taking 7 hours longer to train on the sampled dataset (1.3mn samples). MLP is also more flexible in terms of speed, as reducing the number of layers, epochs, or adding more strict early stopping criteria could all reduce the training time.

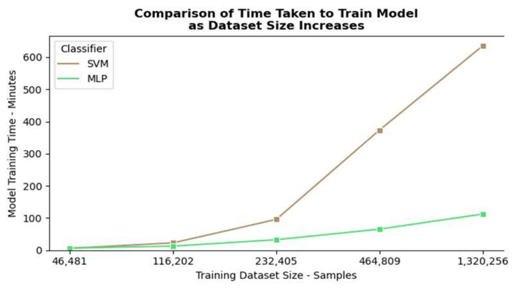


Figure 9: Line plots of time to train per classifier

7. iv) Hyperparameters:

While SVM has comparatively few hyperparameters to search through, the nature of the range of possible values of the regularization parameter C, as well as the polynomial kernel still means that hyperparameter searching is a slow process (average 6 mins per parameter combination with a 10% subset of the data – 46k samples).

MLP has more available combinations of hyperparameters; with many types of activation function and optimizer not tested in the scope of this paper (i.e., RMSprop or Tanh). This can make the model more flexible in comparison to SVM depending on the nature of the problem to be solved.

7. v) Comparison to Reference Papers

Trebar and Steele's SVM performance was ~10% better than my optimal model. The gain is likely due to their approach of using distributed SVM architectures trained on subsets of the data, as well as their use of the SVM^{light} package – an implementation designed for large datasets[2].

The MLP outperformed the ANN in Blackard and Dean's research[1] by 34%, proving the effectiveness of hidden layers for solving non-linearly separable problems, augmented by the architectural developments and range of hyperparameters made available between 1999 and today.

8. Conclusions, Lessons Learned and Future Work

Overall, both models performed well, achieving high accuracies in the classification task. I learnt that there are solutions to the MLP models main drawbacks - using dropout and early stopping to prevent overfitting, or momentum to avoid getting stuck in local minima. Whilst the SVM underperformed in my tests, Trebar and Steele have shown that using distributed SVM architectures, superior performance can be achieved. A further approach could be to combine the two models as Collabert et al.[13] has done, where multiple "expert" SVMs were trained and their estimates fed as inputs into an MLP that acted as a "gater" function to produce predictions. Evaluating and comparing the performance of a random forest model may also prove insightful as they are adept at dealing with high dataset dimensionality.

To improve this work, Bayesian Optimization could be employed when searching for hyperparameters to potentially find more optimal settings, while ROC-AUC curves could be plotted as an improved visual aid to display the performance of each model. I could also experiment with feature reduction techniques targeting some of the more highly correlated variables to observe the impact on model performance and speed.

Lastly, I learnt that sampling methods are not always beneficial for SVM, especially in imbalanced scenarios. Under sampling can remove important datapoints, whilst over sampling can create noise, altering the margin and resulting in worse classification accuracy for the majority class.

References

[1] Blackard, J. A. and Dean, D. J. (1999) 'Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables', Computers and Electronics in Agriculture, 24(3), pp. 131–151. doi: 10.1016/S0168-1699(99)00046-0.
[2] Trebar, M. and Steele, N. (2008) 'Application of distributed SVM architectures in classifying forest data cover types', Computers and Electronics in Agriculture, 63(2), pp. 119–130. doi: 10.1016/j.compag.2008.02.001.
[3] UCI Machine Learning Repository: Covertype Data Set. Available at: https://archive.ics.uci.edu/ml/datasets/covertype (Accessed: 21 March 2021).
[4] Joachims, T. (2002) Learning to Classify Text Using Support Vector Machines. Springer Science & Business Media.
[5] Bottou, L. and Lin, C.-J. 'Support Vector Machine Solvers', p. 27.
[6] Bishop, C. M. and Bishop, P. of N. C. C. M. (1995) Neural Networks for Pattern Recognition. Clarendon Press.
[7] Andrew Ng, Support Vector Machines notes - CS229 Machine Learning: http://cs229.stanford.edu/notes/cs229-notes3
[8] Hornik, K., Stinchcombe, M. and White, H. (1989) 'Multilayer feedforward networks are universal approximators', Neural Networks, 2(5), pp. 359–366. doi: 10.1016/0893-6080(89)90020-8.
[9] Caruana, Rich, Steve Lawrence, and Lee Giles. 2000. "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping." In Proceedings of the 13th International Conference on Neural Information Processing Systems, 381–87. NIPS'00. Cambridge, MA, USA: MIT Press.
[10] Chawla, N. V., K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. "SMOTE: Synthetic Minority Over-Sampling Technique." Journal of Artificial Intelligence Research 16 (June): 321–57. https://doi.org/10.1613/jair.953.
[11] Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. n.d. "A Practical Guide to Support Vector Classification," p.16.
[12] Kingma, D. P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization', arXiv:1412.6980 [cs]. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 23 March 2021).
[13] Collobert, R., Bengio, S. and Bengio, Y. (no date) 'A Parallel Mixture of SVMs for Very Large-Scale Problems', p. 8.

Glossary

Accuracy – The number of correctly identified classes as a proportion of all predictions.
Precision –The proportion of correctly identified positive classes over all positive classes identified.
Recall – The proportion of correctly identified positive classes over all actual positive classes.
Sensitivity – The proportion of positive classes that are correctly identified
Specificity – The proportion of negative classes that are correctly identified.
F1 Score – The harmonic mean of precision and sensitivity.
C – Used after the introduction of soft-margin SVM whereby the parameter 'C' dictates the level of tolerance to give the model when finding an acceptable decision boundary – a higher value of 'C' equates to higher misclassification cost, narrower margin and fewer support vectors[3].
Gamma - A parameter specifying how far the influence of a single training example can reach to affect the decision boundary, a lower gamma value indicating a farther reach.
Learning rate - Dictates the magnitude of the weight updates in the model in response to error values.
Momentum – In addition to learning rate, momentum helps control the step size taken towards estimated minimums and is used to help avoid local minima.
Dropout - Randomly drops neurons, and their connections, from the neural network, it is used to help prevent the network from overfitting.
Activation Function (including Rectified Linear Unit (ReLU)/ Leaky ReLU/ Sigmoid)– A function that is applied in-between hidden layers to transform the output values from the previous layer.
SMOTE (Synthetic Minority Oversampling Technique) - A technique used to oversample by creating synthetic samples of the minority class through a nearest neighbours approach.
Borderline SMOTE[1] – Creates synthetic minority class samples at the borderline between classes where the algorithm thinks there is the greatest chance of misclassification.
Tomek Links[2] – A form of under sampling whereby "borderline" instances from the majority class that the algorithm considers likely to be misclassified are removed.
Oversampling - Creates additional samples, duplicating the minority class at random (with or without replacement).
Under Sampling - Removing samples from the majority class at random.
Decision Tree - A supervised tree-based algorithm
Random Forest – A machine learning algorithm that builds a 'forest' out of an ensemble of decision trees, utilizing majority voting for prediction.
Gradient Descent - Updates model parameters in steps towards a slope gradient of 0.
Stochastic Gradient Descent – randomly selects datapoints at each step to update parameters.
Adaptive Moment Estimation (ADAM)[3] – Utilizes momentum and adaptive learning to converge faster to slope gradient minimum.
Cross Entropy Loss - Compares each predicted class probability to the actual assigned class and penalizes predictions more the further they were from the actual values.
Weight Decay – A regularization technique that adds a small penalty to the loss function to prevent overfitting and exploding gradient.
Softmax - Softmax is the generalized form of sigmoid used in multi-class problems
Epoch – A measure of when the model has updated model parameters based on all dataset samples that have been fed to the neural network in batch form.
Sklearn – A machine learning package in Python.
Bayes Theorem - An equation describing the relationship of conditional probabilities.
Bayesian Optimization – A search technique based on Bayes Theorem that builds a probabilistic model of the objective function to search efficiently for effective parameters.

Glossary References

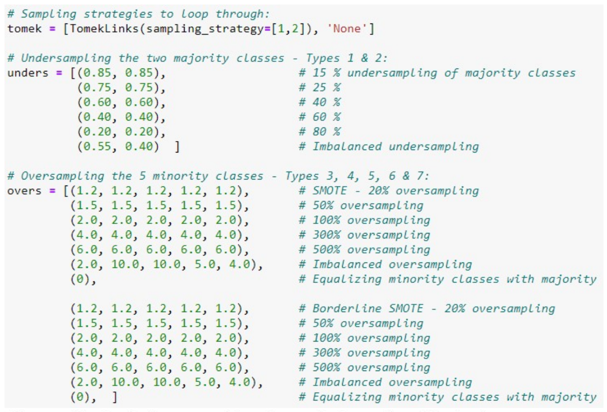
[1] Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning | SpringerLink (no date). Available at: https://link.springer.com/chapter/10.1007/11538059_91 (Accessed: 24 March 2021).
[2] Devi, D., Biswas, S. kr. and Purkayastha, B. (2017) 'Redundancy-driven modified Tomek-link based undersampling: A solution to class imbalance', Pattern Recognition Letters, 93, pp. 3–12. doi: 10.1016/j.patrec.2016.10.006.
[3] Kingma, D. P. and Ba, J. (2017) 'Adam: A Method for Stochastic Optimization', arXiv:1412.6980 [cs]. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 23 March 2021).

Additional Implementation Details

Sampling Strategies Used

Testing of different sampling strategies was performed via a manual grid search with 4-fold cross validation for both models. The range of strategies can be seen in Figure 10. Tomek links and under sampling were applied to the two majority classes, while over sampling was applied to the 5 minority classes.

Equalizing the minority class using SMOTE/Borderline SMOTE meant that synthetic samples were created for every class until each class had the same number of samples as the largest majority class.



```
# Sampling strategies to loop through:
tomek = [TomekLinks(sampling_strategy=[1,2]), 'None']

# Undersampling the two majority classes - Types 1 & 2:
unders = [(0.85, 0.85),                    # 15 % undersampling of majority classes
          (0.75, 0.75),                    # 25 %
          (0.60, 0.60),                    # 40 %
          (0.40, 0.40),                    # 60 %
          (0.20, 0.20),                    # 80 %
          (0.55, 0.40)  ]                  # Imbalanced undersampling

# Oversampling the 5 minority classes - Types 3, 4, 5, 6 & 7:
overs = [(1.2, 1.2, 1.2, 1.2, 1.2),        # SMOTE - 20% oversampling
         (1.5, 1.5, 1.5, 1.5, 1.5),        # 50% oversampling
         (2.0, 2.0, 2.0, 2.0, 2.0),        # 100% oversampling
         (4.0, 4.0, 4.0, 4.0, 4.0),        # 300% oversampling
         (6.0, 6.0, 6.0, 6.0, 6.0),        # 500% oversampling
         (2.0, 10.0, 10.0, 5.0, 4.0),      # Imbalanced oversampling
         (0),                              # Equalizing minority classes with majority

         (1.2, 1.2, 1.2, 1.2, 1.2),        # Borderline SMOTE - 20% oversampling
         (1.5, 1.5, 1.5, 1.5, 1.5),        # 50% oversampling
         (2.0, 2.0, 2.0, 2.0, 2.0),        # 100% oversampling
         (4.0, 4.0, 4.0, 4.0, 4.0),        # 300% oversampling
         (6.0, 6.0, 6.0, 6.0, 6.0),        # 500% oversampling
         (2.0, 10.0, 10.0, 5.0, 4.0),      # Imbalanced oversampling
         (0),  ]                           # Equalizing minority classes with majority
```

Figure 10: Code for searching through Sampling Strategies

Overfitting and Variance Checks – MLP

100 epochs as well as stopping criteria whereby the training stopped if there were 5 consecutive increases in validation cross entropy loss were used. The optimal model was stored after the epoch that resulted in the lowest validation set cross entropy loss. Training and validation losses were tracked to observe any over fitting – see Figure 11 for an example run. The majority of runs stopped before 100 epochs were reached, with an average epoch length of 73.6 across all searches.

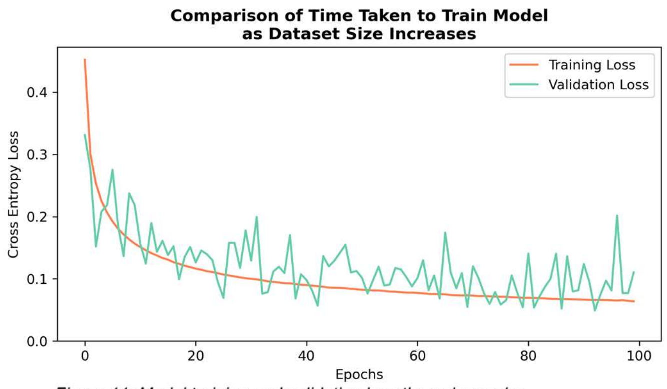


Figure 11: Model training and validation loss through epochs

During grid searching, the cross-entropy loss for each fold was stored so that the lowest scoring model could be inspected to check that the variance of the loss wasn't abnormally high in-between folds. This helped to avoid choosing optimal hyperparameters that might have been affected by anomalies resulting from the randomness of folding or initialization of neural network weights.

Batch Normalization – MLP

Batch normalization was used within my MLP model aiming to increase the speed and reduce the variance of the model. It uses the batch mean and variance to normalize the activation vectors between the hidden layers of the model.

Baseline Comparative Models

In order to construct models to use as baselines in Section 7, I used the same methods as for the optimal models, but with default/basic sets of parameters. The following hyperparameters were therefore used:

- SVC:
    - Defaults from sklearn documentation were kernel = 'rbf', C = 1.0, gamma = 1 / (n_features * X.var()).
    - Source: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- MLP:
    - The same MLP model architecture was used (2 hidden layers), but without any dropout or batch normalisation layers.
    - Learning rate = 0.001, momentum = 0, epochs = 50, optimizer = SGD, activation = ReLU, max hidden neurons = n_features.