1 Introduction

- **Group members**
  Zhewei Chen, Milad Taghavi, Yan Qi Huan

- **Team name**
  TheMagicMarijuanaForest

- **Division of labour**
  Equal contribution from each member

2 Overview

- **Models and techniques tried**
  - **SVC:** SVC with radial basis function, linear, polynomial & sigmoid kernels
  - **Decision Trees:** Gradient boosted and Adaboost decision trees
  - **Neural Networks:** Various different layers, dropout probability, and bagging

- **Work timeline**
  - **Week 1:** Focused on finishing hw4, finding teammates, discussed structure of training data, and initial strategies we would try.
  - **Week 2:** Submitted initial results to kaggle, met and discussed alternative approaches to improving model accuracy, implemented strategies independently and then met up to discuss approaches to combine and refine each approach.

3 Approach

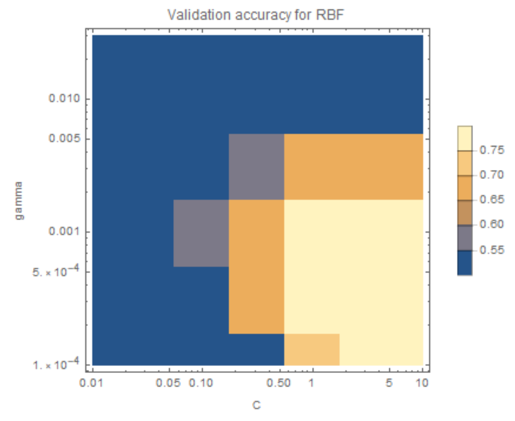
- **Data processing and manipulation**

  We checked the training data for repeated samples and removed them using numpy.unique(). The data was shuffled and normalized using sklearn.normalize() and sklearn.shuffle(), which scales the input vector into a unit vector of N dimensions in L2 norm with zero mean and randomizes the order of the input data. For neural networks, batch normalization was performed before each activation layer. PCA was also used to reduce the feature size of the input data, but was found to hurt model accuracy and hence it was not used.

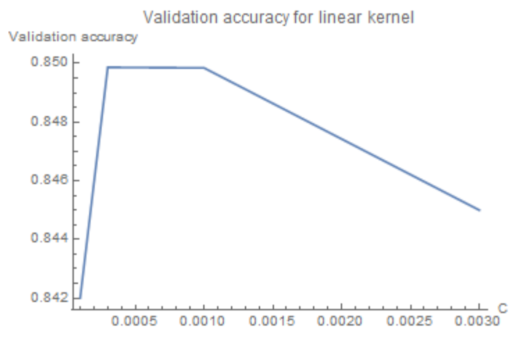
- **Details of models and techniques**
  - **SVC:** The first model that we tried was using SVCs (Support Vector Classifier), the classification variant of SVMs. We chose SVCs as an initial trial model because it was easy to implement with feature normalization as the only preprocessing step.

Additionally, since the final trained model only depends on the support vectors and not the entire dataset, we felt that this would be helpful as dimensional reduction in preventing overfitting since the model would only learn the features of the margin from the relatively smaller number of support vectors instead of using every single training datapoint.

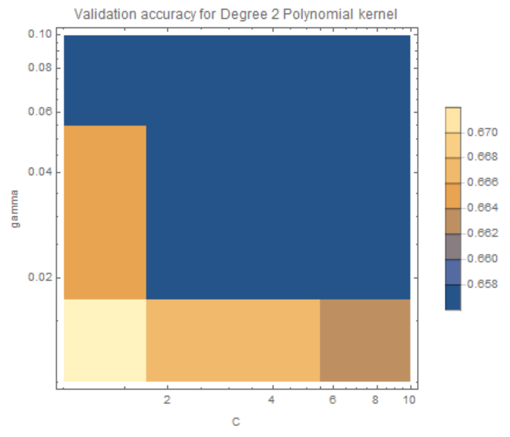
The tunable parameters for the SVC model were C (regularization parameter), kernel type (we tried radial basis functions, polynomial, linear, and sigmoid), gamma (width of the fitting function), degree (only for polynomial) and coef0 (an optional constant offset for non-linear models). For speed of the grid search over parameter space, we first split the data to obtain a 10% subset of the data which was much faster to train and cross-validate on. Using this, we obtained a rough range of optimal parameter values, and we obtained the true optimal parameter values by repeating parameters near the peak on the full dataset.



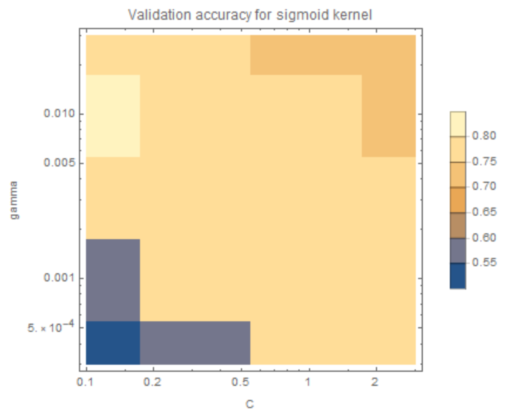
(a) Sweep over parameters for RBF kernel



(b) Sweep over parameters for linear kernel



(c) Sweep over parameters for degree 2 polynomial kernel



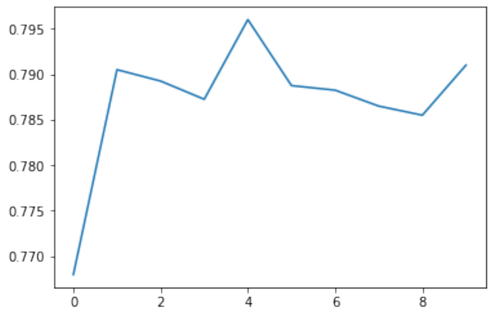
(d) Sweep over parameters for sigmoid kernel

Figure 1: Cross validation error across hyperparameters for various kernels for SVC

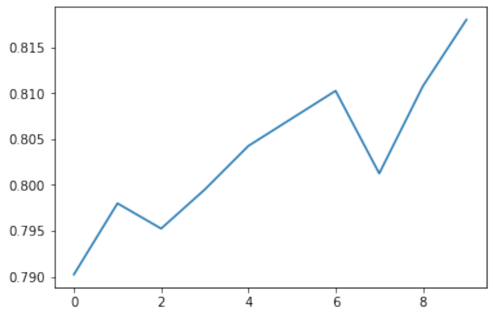
Following the parameter searches, the optimal model that was chosen was $C = 4.0, \gamma = 0.0002$ for the RBF kernel, which had a **validation score of 0.85125** and a training score of 0.910. However, this was only enough to get a midddling score on the leaderboard and thus we tried other methods.

– **Gradient Boosted Decision Trees:** The bag of words data set was thought to have nonlinear interactions between each of the features which influence the final classification. Unlike linear models, decision trees could better fit these nonlinear interactions.

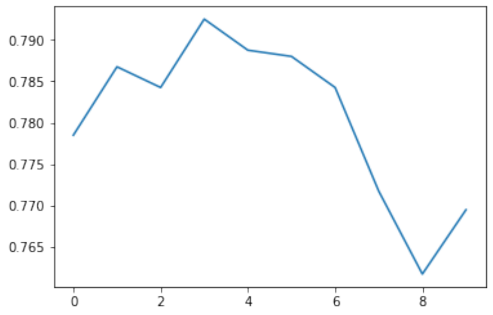
Various parameter optimization were done to achieve the highest accuracy on cross-validations with 10 folds splitting of the input data. Figures below show the parametric sweep for max tree depth, learning-rate and number of estimators. In each optimization, other parameters were kept fixed at a reasonable value to reduce learning times by removing the need for an exhaustive grid search.



(a) Sweep over max tree depth from 1 to 10



(b) Sweep over number of estimators from 100 to 1000



(c) Sweep of learning rate from 0.1 to 1

Figure 2: Cross validation error across parametric sweeps of multiple parameters

As shown above, the optimum learning-rate and max tree depth are, 0.3 and 4 respectively. Also, as shown in figure 2b, the accuracy is almost monotonically increasing with an increasing number of estimators. To limit the learning time, 500 estimators was chosen as the final value. The maximum achieved accuracy in **cross-validations was found to be at 0.849** and the training time was about 30 minutes. The advantage of this model was its simplicity, but it took a long time to train the model and the accuracy was found to be lower than other classifiers.

– **AdaBoosted Decision Trees:** AdaBoost was tried as an alternative to gradient boosting because the algorithm was designed for feature classification whereas gradient boosting was designed for decision tree regression. Ensemble tree numbers from 100 to 1000 were tried. Although training error decreased with ensemble tree number, **validation accuracy plateaued at 0.83**. Training time also lengthened significantly (>2min) when ensemble tree number was greater than 500.

– **Neural Networks:** Neural Networks were tried last as an alternative to SVMs and Decision Trees as it was the only thing left we had not tried from class. Softmax activation on 2 neurons was used as the final output layer so we could get a probability on sentiment classification. The most likely classification was threshold to 1 while the other to 0 and the output was un-onehot encoded to create the final output.

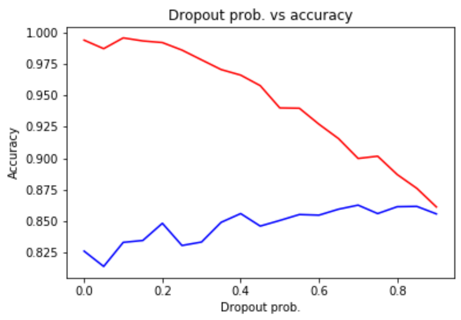
  Initial results with a 3 layer dense network of neurons showed good fit to the train data (less than 0.01 training error). The nonlinear interactions between each feature could be well approximated by neural networks more so than SVM and decision trees. Neural networks also took less time to train than decision trees (<60s per epoch) while getting good fits (less than 0.15 validation error). However, like the SVM classifier, **validation accuracy plateaued at 0.845** regardless of neuron network depth or width.

  The dropout probability on the first activation layer was optimized as a hyperparameter using cross-validation. By choosing the optimal value, the validation accuracy was substantially increased. We found that increasing the dropout probability raised validation accuracy and decreased training accuracy until prob = 0.4. Thereafter, training accuracy and validation accuracy both declined in performance, indicating that underfitting had likely occurred.

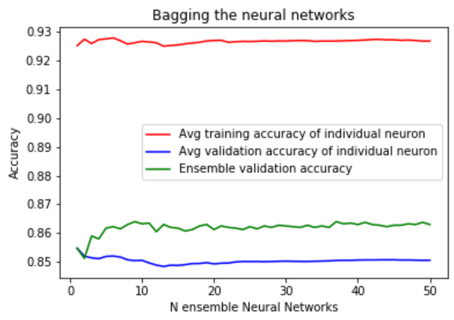
  Although neural networks could fit training data extremely well, it suffered from high model variance as we expected due to the high number of tunable parameters in the architecture. This was indicated by the divergence between training accuracy and validation accuracy. This also explained why increased dropout probability improved validation accuracy, as it helped regularize the model against overfitting.

  In fact, from the divergence between training and test error, we learned that the primary obstacle in the way of each learning model that we had tried so far was overfitting. This indicated that the training data was probably undersampled from the true probability distribution. Each of our models could overfit the training data easily despite our best efforts to maximize validation error through hyperparameter optimization.

  To address this underlying variance problem, we decided to try two different approaches: a voting model as well as bagging of Neural Networks.



(a) Hyperparameter optimization for neural networks

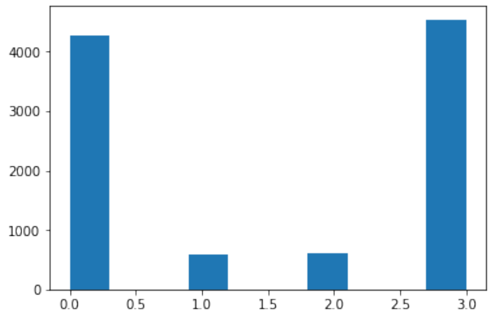


(b) Bagging the neural networks

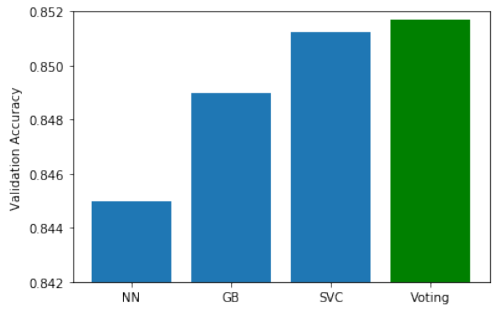
Figure 3: Neural network optimization

- **Voting Model: Combination of SVM, NN and GB:** We first tried a merger of all three of our models: Support Vector Classifier (SVC), Neural Networks (NN) and Gradient-Boosted decision trees (GB). All three models were trained on the same data and the final prediction was a majority vote of each model's prediction. This was inspired by the methods of bagging and random forests, where we train many different models and average their predictions. In this case, since the output is simply a binary output, we use majority voting instead of averaging.

  As expected, this model was able to increase the test accuracy by reducing the variance of each individual model and hence reducing the effects of overfitting.



(a) Distribution of number of votes for each test point. The low number of points with 1 and 2 votes indicate that the 3 models agree with each other most of the time.



(b) Improvement in the validation accuracy by using the voting of the 3 models

Figure 4: Voting model utilizing the SVC, NN, and GB models in order to reduce variance.

As can be seen from Figure 4b above, we obtain a small but noticeable improvement in terms of the validation error when compared to each of the individual models. Figure 4a also confirms that the voting is intuitively making sense since most data points have either all models voting for against it, and hence our models are mostly in agreement.

The **validation error obtained using this model was 0.8517**, compared to 0.845, 0.849, and 0.85125 respectively for the NN, GB and SVC models. We submitted this for verification on the test set and obtained a **test error of 0.85640** which was our highest score thus far. This therefore was a strong indication that such an approach would be fruitful.

– **Final Model: Bagging of Neural Networks:** Next we tried just bagging the neural networks. 10 to 50 neural networks of the same architecture were trained on the same data. The final output was an average of these neurons and this gave us the best public score on the kaggle test set. We dub this methodology as the ”Random Marijuana Forest” because THC affects neurons instead of decision trees.

4 Model Selection

- **Scoring**
  The method of scoring each of the trained models was simply via 0/1 binary loss, since the problem inherently only involves predicting 0 or 1 as the outcome. It was found that most models could achieve much higher training accuracy than validation accuracy, with training accuracy at least 85% for pure AdaBoost and approximately 99% for NN.

- **Validation and Test**
  We used 5-fold and 10-fold cross validation accuracy as our scoring function. The lowest validation accuracy was achieved with pure AdaBoost at 83%, and the highest accuracy achieved with NN (with bagging) at 86%. As such NN (with bagging) was chosen as the final classifier.

5 Conclusion

- **Discoveries**
  We had some interesting discoveries along the way, including:

  – We had hypothesized that PCA might be helpful in terms of reducing the number of dimensions of the data and thus allowing us to reduce training time while keeping most of the information encoded in the data. However, we eventually abandoned it as it almost always led to a reduced training and validation accuracy. A possible reason for this is that the appearance of words in the list are mostly uncorrelated, and thus there are not many correlated dimensions for the PCA algorithm to reduce.

  – As we already knew, we found that preventing overfitting was the primary obstacle preventing us from getting a better out-of-sample performance. In fact, by just using the in-built sklearn packages, it is extremely easy to get > 0.9 training accuracy, and it is much easier to get a good in-sample performance than to extend it to test data. Accordingly, we spent most of our time trying to implement more or different kinds of regularization as opposed to trying to get better training error.

- **Challenges**

  - We spent about half the time in this project finding the right model, and we went through a total of 4 main architectures: SVC, Gradient-Boosted decision trees, AdaBoost decision trees, and Neural Networks. Part of the challenge was trying to use our intuition about what methods would work well to decide what model to start working on first. However, as we initially predicted that NN would lead to the most overfitting, we decided to leave it aside until we had tried all the other methods, and this was somewhat lost potential since NN turned out to be the best performer in the end after some tuning. Therefore, while it might be good to use one's instincts, it might also be good not to rule out any particular architecture too early.
  - Optimizing the hyper-parameters also turned out to increase the training time significantly. This was because by doing a grid search over some possible range of parameters, we would need to multiply the same training time by several tens of times, and this meant that a training time of 10 minutes would become several hours. Some solutions we tried included doing grid search on a subset of the data as well as searching 1 parameter at a time instead of a multi-dimensional search. Other ideas included using adaptive searching to start with a coarse grid and then adjust the searched parameters based on the outcome.

- **Concluding Remarks**
  ”Random marijuana forests”, or rather, training a large number of models with low bias and high variance and then averaging them, turned out to be the critical component in our approach towards solving the bias-variance trade off. Other extensions in the future could be to try doing this “random forests” ensemble approach with other models other than neural networks as the base classifiers.

  It was fun. ☺

  See link for team source code

  https://github.com/miladtaghavi/CS155Mini1