1 Introduction

- **Group members**
  Yukai Liu, Botao Hu and Jian Xu
- **Team name**
  aaaalbert
- **Division of labour**
  Yukai Liu implemented XGBoost algorithm, AdaBoost algorithm and Logistic Regression algorithm. And Yukai Liu was in charge of model stacking and selection.
  Botao Hu implemented Neural Network algorithm and K-Nearest-Neighbors algorithm.
  Jian Xu implemented SVM algorithm and Random Forest algorithm. And Jian Xu used PCA to process data in parts of our experiments.

2 Overview

- **Models and techniques tried**
  - **XGBoost:** We used the open-source XGBoost library, and achieved the best validation accuracy 85.5%.
  - **AdaBoost:** We used AdaBoostClassifier in sklearn, and achived the best validation accuracy 84.7%.
  - **Logistic Regression:** We used Logistic Regression in sklearn, and achived the best validation accuracy 84.9%.
  - **Neural Network:** We used the Sequential model from Keras library, and achieved best validation accuracy of 85.6%.
  - **K Nearest Neighbors:** We trained KNeighborsClassifier object from sklearn library, and achieved best validation accuracy of 71.4% with PCA pre-process.
  - **SVM:** We used svm.SVC, resulting in a cross validation accuracy of 84.8%.
  - **Random Forest:** We used RandomForestClassifier in sklearn, resulting in a cross validation accuracy of 84.7%.
  - **Techniques: PCA:** We tried to implement PCA on dataset before training, but it turned out not to be helpful in the cross validation and the final testing score.
  - **Techniques: Model Stacking:** We stacked all 7 models together, and used logistic regression to get final model. This technique improved the cross-validation accuracy of the best model by 0.2%.
- **Work timeline**
  - **Week 1:** We tried different models to see which kinds of methods would work best, and made these models ready for stacking.

– **Week 2:** We stacked all 7 models together to get the best results. We found Neural Network and XGBoost worked the best in this problem, so we spent more time training and tuning these two models. We also tried to use PCA to process data and trained our models again, but it didn't work very well.

3 Approach

- **Data processing and manipulation**

  PCA: We tried to use Principle Component Analysis (PCA) to process the data before training. PCA mathematically decomposes the features onto a set of principle components that are linearly uncorrelated. After the decomposition, the first k largest principle components are chosen to fit in the following training models. In principle, it is a kind of regularization method that can overcome a part of overfitting. The reason that we tried to use PCA is that we found our models could achieve high training accuracy (over 90%), but validation accuracy was bottle-necked by about 85%, and we suspected that overfitting might play a role on this. We tried some regularization methods that are related to models themselves, but they did not improve the results much. Finally, we thought of manipulating data and tried PCA.

  We used PCA from sklearn.decomposition in our data processing. We tried to decrease the number of components in the range of 200 to 800, and for each number of components we used the processed data to train all of our models and finally bagged the trained models. It turned out that PCA did not help improve the validation accuracy. Both training accuracy and validation accuracy decreased after PCA. Some models, such as Support Vector Machine, is not sensitive to the number of components that PCA gave, and the validation accuracy decreased a little (about 1%) when we set the number of components as 500. Some models, such as Random Forest, is sensitive to the number of components that PCA gave, and the validation accuracy decreased a lot (about 10%) when we set the number of components as 500.

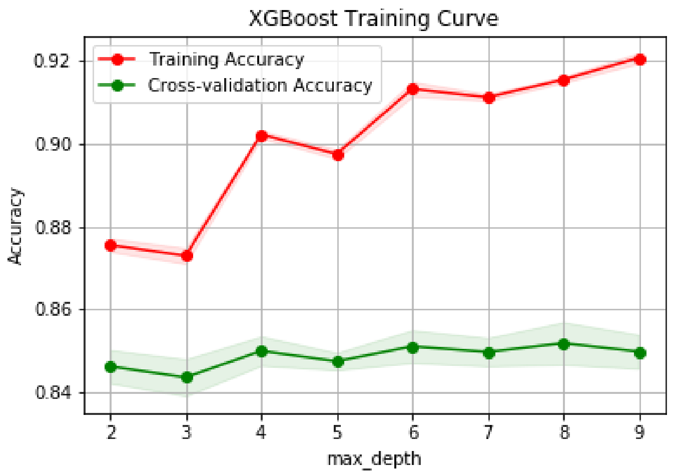
  For our final model, we did not use PCA to pre-process the data.

- **Details of models and techniques**

  1. **XGBoost:** We decided to try this model because it was very popular in Kaggle recently. It was basically an advanced version of Gradient Boosting Algorithm. Compared with Gradient Boosting, on the one hand, it uses second order derivatives to better optimize loss, on the other hand, it explicitly includes l1 and l2 regularization of leaf nodes in its loss function to prevent overfitting. Furthermore, it learns column subsampling from random forests, which also helps decrease overfitting. In terms of disadvantages, first of all, this model has lots of hyper-parameters, so that it may be very time-consuming tuning these parameters. Then, in this project we use decision tree as the base classifier, so as an ensemble method, this model has its strength limit, it may not be powerful enough to capture some more sophisticated structures in data.

     We used the open-source XGBoost library in our implementation. In our experiments, we first used a relatively high learning_rate, 0.1, and set other parameters to some initial values, max_depth = 5, min_child_weight = 1, gamma = 0, subsample, colsample_bytree = 0.8, and used cross validation to find a proper n_estimators. Then we fixed learning_rate and n_estimators, and did grid searches for parameters max_depth ranging from 3 to 10, min_child_weight ranging from 1 to 6, gamma ranging from 0 to 0.4, subsample, colsample_bytree ranging from 0.6 to 0.9.

In all these parameters, we found max_depth to be the most important one. It controls the capability of this model. The following graph shows the training curve with different max_depth. We can find that as max_depth goes up, training accuracy increases, and cross-validation accuracy also increases. We choose the parameter with the highest cross-validation accuracy. From this graph we can also see that XGBoosting does not suffer from over-fitting.



After we found the best hyper-parameters with a proper bias variance trade-off, we lowered learning_rate and found the final n_estimators. Final cross-validation accuracy of this model is 0.8551. The optimized hyper-parameters are as follows:

```
max_depth = 8
n_estimators = 446
min_child_weight = 5
l1_reg = 0.01
l2_reg = 0.01
subsample = 0.6
colsample_bytree = 0.7
```

2. **Adaboost:** We tried this model because it is a boosting model for classification. Its advantage is that it reduces bias of weak classifiers by training on reweighted data, and by many models voting together, it doesn't suffer from overfitting severely. Its disadvantage is that it runs very slow in our experiments and it is not a very strong model.

   We used the built-in methods from scikit-learn in our implementation. We experimented with parameters learning_rate, max_depth and n_estimators in the model. We first found a proper learning_rate 0.1, then fixed it and did a grid search for max_depth ranging from 3 to 6 in increments of 1, and n_estimators ranging from 500 to 2000 in increments of 300. The final cross-validation accuracy of this model is 0.8468 with max_depth = 5 and n_estimators = 1500.

3. **Logistic Regression:** We tried this model because it is simple and is very effective for nearly linearly separable data. Its advantage is that it is simple and is one of the best linear models. Its disadvantage is that it is not a very strong model. We used the built-in methods from scikit-learn in our implementation. We experimented with parameter c, which stands for inverse of regularization strength in the model. Ranging the parameters between [1e-4, 1e4], we got scores in the range of 70% to 85%. The final cross-our ensembled modelvalidation accuracy of this model is 0.8492.

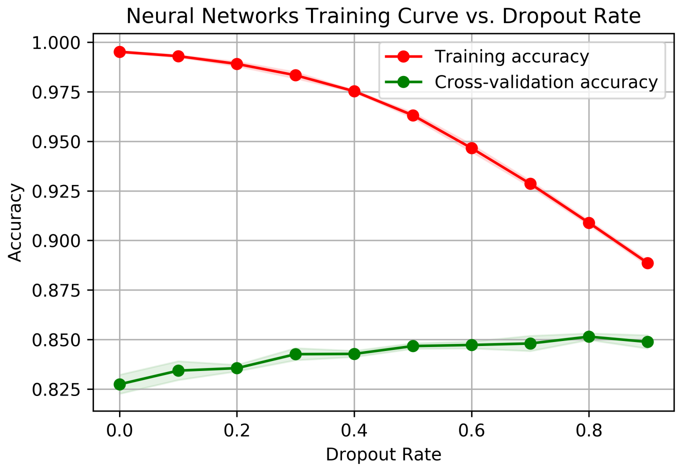
4. **Neural Networks:** We tried Neural Networks as one of our primary models. From this link we learned that the author, using the same bag-of-words representation of data, built up a neural network to analyze movie reviews. The author tried one hidden layer with 50 neurons, and achieved 91% test accuracy. Since the author was analyzing the same topic as what we have in Kaggle competition, and got relatively good results (in comparison with the leaderboard). What's more, a single-hidden-layer structure is easier for parameter adjustment. So, we decided to start with the author's structure and used hyper-validation to optimize parameters.
   The structure of Neural Networks is as the following (we used open source Keras library for NN implementation):

```
model = Sequential()
model.add(Dense(num_hidden_x, input_shape=(len(X_train[0]),)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(dropout))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=Adam, metrics=['accuracy'])
fit = model.fit(X_train, y_train, batch_size = 32, epochs=num_epoch, verbose=0)
```

   After testing different parameters such as `dropout`, `num_hidden_x`, `optimizer/learning_rate`, and `num_epoch` using cross-validation with 5 folds, we came to the optimized parameters as:

```
dropout = 0.8
num_hidden_x = 70
learning_rate = 0.001
optimizer = Adam
num_epoch = 10
```

   Here are some thoughts in hyperparameter validation process:
   Dropout rate: this rate stands for regularization power. As known to all that Neural Networks tend to have overfitting issues since it can fit the training data very well using non-linearity. In this competition, overfitting is also an important factor which partially determines the performance of test accuracy on the leaderboard. As a result, here we used a very strong regularization, and limited the training accuracy within 91% and 92%, where we got best validation accuracy. The following graph shows the training curve with different dropout rates.



# Neurons in the Hidden Layer: we started from 50. We believed that this number should be relatively small since the lower model's complexity is, the less likely we would encounter overfitting issues (unfortunately we were wrong). So, we tried this parameter from 10 to 100, and found that there is a peak of validation accuracy at 70.
Learning Rate: this rate is like the step size of gradient descent. It turned out that the default learning rate is the best one: lower rate caused incomplete descent; higher rate caused non-convergence.
# Epochs: In adjusting other parameters, we typically ran the model for 5 epochs unless there was no sign of convergence in the accuracy log. The Neural Networks model is not time consuming in this case (each epoch usually costs for less than 10 seconds, thanks to the simple structure we started with). After that, we tested different number of epochs, and found that the accuracy converges at around 7 to 15 epochs, and goes down afterwards. So, we chose to train for 10 epochs to avoid underfitting and overfitting.
Under this approach, we achieved an average validation accuracy of 85.25% among 5 folds, with the maximum accuracy of 86%. In the submission, we also achieved relative good results: 85.48% in private and 85.52% in public. This also tells us we had no overfitting issues.

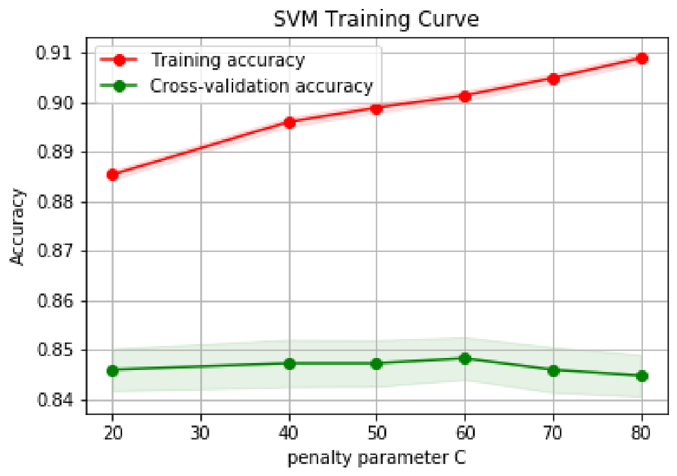
At first, we were quite satisfied with the result and believed that this neural network structure is among the most optimized ones, so we started to try other models as well as bagging strategy. However, at the last night before competition ends, we were informed by the TAs that Neural Networks can actually achieve higher accuracy with multiple hidden layers and larger number of neurons.
Therefore, we inserted 3 more hidden layer with the same structure as the first one, fixed all the other parameters and only tested different numbers of nerons and dropout rates for each layer. We started from `dropout = 0.5` and `num_hidden_x = 200` for each layer. Unfortunately, due to the limit of time we end up with

```
dropout = [0.65, 0.55, 0.45, 0.35]
num_hidden_x = [200, 200, 200, 200]
```

and avarage validation accuracy of 85.6% among 5 folds, which was already larger than the previous one. Though the test accuracy in submission is around 85.5%, this model suggested that multi-layer Neural Networks have a good potential. Sadly we didn't have time to explore it.

5. k-**Nearest Neighbors:** We tried this model in order to explore the clustering property of the data. We trained `KNeighborsClassifier` object from the sklearn library, and used cross validation to test its accuracy with different numbers of centers. We found that this model is time-consuming and inaccurate. If we preprocess data with PCA strategy, we could reach best validatioon accuracy of 71.4% when $k = 50$; without PCA, the best accuracy would not exceed 70%. So, we decided not to use this model individually; instead, we used it as a reference in bagging process.

6. **SVM:** We used support vector machine (SVM) as one of the classifiers. Since the data has high dimensions (1000 features), we expected that it provided enough degrees of freedom for the dataset to be approximately linearly separable. The advantage of using this model is that it is simple. The disadvantage is that it does not have many parameters to tune in order to get higher accuracy. We used the built-in methods SVC from sklearn in our implementation. After trying different kernels, we found rbf is the best for cross validation accuracy. By varying the penalty parameter C from 0 to 100, we got the classification accuracy being above 80%. When C=60, we achieved the highest cross validation accuracy about 84.8%. The following figure shows the learning curve with different penalty parameter C.

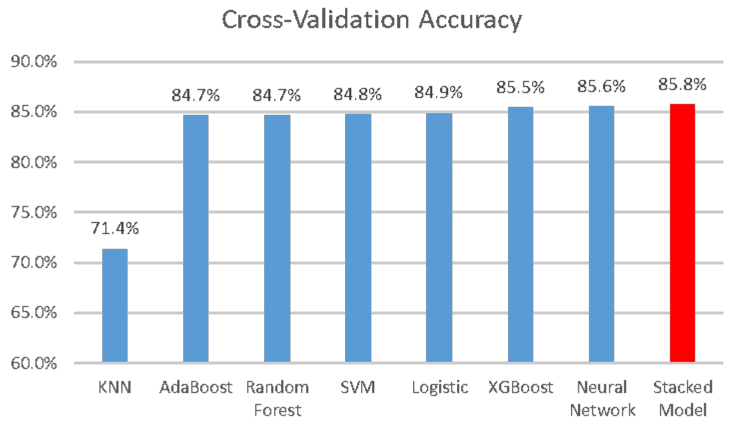


7. **Random Forest:** We used random forest (RF) as one of the classifiers. In principle, the random forest algorithm applies bagging to decision tree learners after randomly picking up samples and features in decision tree learning processes. Usually it can reduce some of overfitting that happens in the decision tree algorithm. Here, we used the built-in methods tree and RandomForestClassifier from sklearn in our implementation. The parameters that mainly determine the model performance during the training processes are max_depth , max_features and n_estimators. Since the number of features is quite large, before limiting the max_features number, the model suffers from strong overfitting. By setting the max_features as log2, which means the maximum number

of features is about log2(1000)=10, we tuned the max_depth of the model. By varying the max_-depth from 10 to 150, we found the optimal max_depth is 90 for minimizing the cross validation. N_estimators is the number of the trees during the training process. We tuned n_estimators from 100 to 1500. Actually when n_estimators is large enough (¿750), the model performance converges and no longer improves any more. Therefore, to accelerate the training process, we chose n_estimators=750 during our training process. When setting max_depth as 90, max_features as log2, n_estimators as 750, we got the best cross validation accuracy about 84.7%.

8. **Technique: Stacking:** We tried this technique because ensembling different models can usually improve final results. Stacking is a model ensembling technique used to combine information from multiple predictive models to generate a new model. Its advantage is that the stacked model will outperform each of the individual models due its smoothing nature and ability to highlight each base model where it performs best and discredit each base model where it performs poorly. For this reason, stacking is most effective when the base models are significantly different. Its disadvantage is that selecting models from validation set may overfit validation set.

   The procedure of stacking is that we first trained different models separately and got the best hyper-parameters for each model. Then we divided training data into five parts. For each part, we trained each model on the other four farts and made predictions on the last part. We used these predicted results as our new features, since we had 7 models, we would have 7 new features for each data. We then trained a new model based on these 7 new features. For the output model, we have tried SVM, logistic regression and random forest. We found that logistic regression had the best result. The final output of the stacked model improved accuracy of our best single model by 0.2-0.3%.



4 Model Selection

- **Scoring**

  We scored models by their validation accuracy, but we didn't rule out any of them. Instead, we used all of them in our ensembled model, in order to boost up accuracy and avoid overfitting. In this way, we used validation error to approximate test error, and directly minimized validation error while not worrying about bias/variance decomposition.

- **Validation and Test**

  We tested different models such as SVM, logistic regression and random forest for our ensembled model, using classifications of single models as features. It turned out that logistic regression is among the best ones. In cross validation, our ensembled model improved accuracy of our best single model by 0.2-0.3%. From the scoreboard we know the test accuracies of this model reached 85.78% in public, and 85.74% in private, which were satisfying and showed that this model did not overfit the public scoreboard.

5 Conclusion

- **Discoveries**

  1. Neural Networks tend to have a better performance in this data set. We think this model has good potential since its non-linearity and multi-parameters help the model to learn better.
  2. Generally, PCA does not help to improve the accuracy for single models in this data set. Only KNN model had positive response to PCA strategy. So, we used PCA as a reference in bagging.
  3. Model ensemble is a useful strategy to boost up accuracy. Our final model with stacking reached the highest accuracy. Also, stacking helps to prevent overfitting, since in the private board, our ranking goes up a little bit (public board: 25th, private board: 22nd).
  4. We found out that many models had similar validation accuracies, and the performance of these models are partly determined by the properties of the dataset. That is to say, NN might perform well in this dataset, but in other learning scenarios, models like SVM or XGBoost could stand out.

- **Challenges**

  From the Kaggle leaderboard one can easily tell that the most obvious challenge of this competition is limited accuracy. There are several reasons.

  1. Sentiment is vague. We thoroughly examined the data set and found that some data full of words like ”love”, ”like”, or ”enjoy” are labeled as bad reviews. In other words, even ”human learning” cannot save us.
  2. Missing information. Infrequent words are cut off from the data, making labelling even harder. So, in the bagging process, we often saw some data where labels of all our models are against the ground truth.
  3. No explicit distribution patterns. We tried to analyze the distribution of data. We used PCA and KNN to explore its geometric distribution; we even calculated the length of each review as a new feature, but it seemed irrelevant with the sentiment inside.

Another challenge for our team is that we were unaware of the potential of Neural Networks. The overall strategy of our team is that each team member train a bunch of models, and combine these models to boost for better results. However, we underestimated the accuracy of NN and ended up with a weak primary NN model. Stacking did help us, but only helped a little bit since our primary models are not so accurate.

- **Concluding Remarks**

  Though a bit unsatisfied with final ranking, we enjoyed this competition. We had a hard time picking up classifiers, validating parameters, and waiting for accuracy results of complex models; but it was fun doing "human learning" on data, ensembling models, and checking leaderboard everyday (especially at the end of competition). From this project, we gained a more practical insight of machine learning, as well as a deeper understanding of different learning models. We believe we can do better next time.