[redacted] Supervised Learning

By **Sahil Gupta** [redacted]

I. Abstract

- To understand how supervised learning models work & how they can be applied for classification & prediction, I choose 2 unique datasets derived from **UCI ML archive**. Both the datasets involve *binary classification* and are non-trivial as showcased by fact that several experiments were run to build performant models and AWS ML instances were used to build models in reasonable time.
- **Scikit Learn** was used throughout the assignment because of its detailed & thoughtful documentation and powerful capabilities such as great out of the box ML Classifiers, parameter tuning via GridSearchCV and supporting tools such learning curve, validation curve and ROC curve plotting functions. I've borrowed code directly from their website and cited it in-line with source code wherever used. I also used the book "*Introduction to Machine Learning with Python*"[1] written by authors' of Scikit Learn, Sarah Guido & Andreas Müller to employ techniques taught there.
- **Classification Problem I: UCI Adult Census Dataset**[2]
  - The goal of this classification problem is to predict whether income exceeds $50K/yr based on 1994 census data. This dataset contains 32561 instances with ***14 attributes each (multivariate) which are either categorical or integer (heterogeneous)***. Each entry contains demographic attributes about an individual such as age, sex, occupation, race, workclass, education, marital-status, relationship, etc.
  - The ***large dataset size of ~32,000 by 14*** *makes this very useful to analyze performance of machine learning algorithms.* This problem is clearly non trivial as experimentation requires significant tuning to make it performant.
  - The final column (also known as target value) contains if an individual makes more than 50,000$ per annum.
  - This problem is very interesting especially in 2020 (which is a census year!) because one can clearly see what factors play a role in an individual's income. There *are several commercial applications such as targeted advertising, understanding factors behind an individual's credit history, etc. Government agencies can use these insights to build targeted social programs to remove systemic problems in communities. E.*g. If we learn that a female gender or person of "color" has lesser income due to biases, this data can be used to make valuable future decisions in these communities & organizations to have a more equitable income distribution.
  - This ***dataset is imbalanced*** because there are 3 times more rows with <50k USD income than >50k USD income.
- **Classification Problem II: UCI Spambase Dataset**[3]
  - The goal of this classification problem is to predict whether an email is spam (unsolicited) or non spam based on data collected from HP Labs postmaster & individuals who filed spam along with work & personal emails of authors. This dataset contains *4601 instances with* ***57 attributes each (Multivariate) which are Integer & Real only***. Each row contains either percent of words in email that match a "Word" or "Character
  - The final column contains whether an email was considered spam (1) or not (0).
  - This dataset of size ***4,600 by 57 is a stark contrast from the census dataset above because it doesn't contain any categorical features & is much smaller in size***. While the number of features is much higher, it's much easier to process since it's integers between 0 & 1 ***(homogeneous)***. Categorical census data needs to be processed before using it for ML models.
  - This problem interests me because *email spam detection is non-trivial & spam is costly (20 billion USD/annum*[4] *& Appian, where I work, just faced a major e-mail spam attack from Russian sources via Ryuk Ransomware*[5]*). Moreover, this dataset's differing nature from the Adult Census data will show us how several ML models compare.*
  - This dataset is ***slightly imbalanced*** because there are 1.5 times more non spam email instances (61%) than spam (39%). This imbalance is unlikely to cause issues but we'll still use some intelligent scoring techniques.

II. Feature Engineering & Preprocessing

Adult Census Dataset

1. The dataset is analyzed using missingno visualizations[6] to find missing attributes. Those were dropped from the dataset as those were creating challenges for ML models & reducing their efficiency.
2. *Binning*[7,8] is performed on continuous features: `age` & `hours.per.week`. This helped discretize continuous values.

```
age_bins = pd.interval_range(start=0, freq=10, end=100, closed='left')
adult_census_data['age_binned'] = pd.cut(np.array(adult_census_data['age']), bins=age_bins,
labels=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).codes
```

3. Some features as age (0-90), education number (1-16) & hours per week (0-100) still had large value ranges. Hence, they needed to be *scaled/normalized*[9,10,11] to a float between 0-1 using `sklearn.preprocessing.minmax_scale`.

```
adult_census_data['age_binned_normalized'] = minmax_scale(adult_census_data['age_binned'])
```

4. *One hot encoding*[1] of *workclass, occupation, marital.status, race & sex* is performed to convert categorical string data into discrete numbers 0 & 1. This greatly improves performance of ML models as numbers are easier to process.

```
adult_census_data = pd.get_dummies(adult_census_data)
```

5. Each feature was analyzed for categorical data distribution against income by using histograms, manual inspection & printing % distributions. This showed most valuable features for predicting income & ones that can be dropped to reduce feature space e.g. `'fnlwgt', 'relationship', 'capital.gain', 'capital.loss', 'native.country'`. Some other features were dropped too after they were normalized, binned & one hot encoded.

```
adult_census_data = adult_census_data.drop(columns=['fnlwgt', 'relationship', 'capital.gain',
'capital.loss', 'native.country', 'education', 'age', 'age_binned', 'hours.per.week',
'work_hours_binned', 'education.num'])
```

Spambase Dataset

1. This dataset was much easier to work with as most attributes ranged between 0 to 100 (%) real continuous numbers.
2. It only required normalization using `minmax_scale` for *word and character frequency* attributes since scale matters.
3. I decided to standardize capitalization associated attributes ('capital_run_length_longest', 'capital_run_length_total') i.e. ones where email contained a large number of capital letters since they were spam. Such data is uniformly distributed with some outliers that skew the results and hence needed *standardization*[11] (instead of normalization).

```
spam_capitalization_features = spam_base_data.iloc[:, -4:-1]
spam_capitalization_features_standardized =
pd.DataFrame(StandardScaler().fit_transform(spam_capitalization_features))
```

III. Supervised ML Algorithms Analysis and Trade-offs

Cross Validation (CV)

First the data was split using `sklearn.model_selection.train_test_split` default test_size=0.25, shuffle=true and stratify=y i.e. 25% test data and 75% training data randomly split while maintaining target's distribution balance. Then, `sklearn.model_selection.StratifiedShuffleSplit` CV was used with 3 n_splits (more splits led to smoother graphs but very large run times) and test_size=0.3. This uses stratified randomized folds made by *preserving the percentage of samples* for each class. This meant a validation set of 22.5% and training set of 52.5% (total 75%). CV gave *more accurate scoring of performance of out of sample (test) scoring and much greater choice of metrics* as we'll see below. It also *allowed for use of all examples/instances for training & testing data* while comparing models.

Metrics Chosen

1. **Weighted f1_score**[12]: Both the problem sets, while very different binary classification tasks, have imbalance. Hence, f1_score is the chosen metric over default 'accuracy'. f1_score is calculated per target class and equals `2 * (precision * recall)/(precision + recall)` where precision is "fraction of relevant instances among retrieved instances" and while recall is "fraction of the total relevant instances that were actually retrieved". Moreover, weighted f1_score seems best because it takes into account the number of true instances for each target class. E.g. Adult census data has 3 times more rows with <50k USD income (say class 1) than >50k USD income (say class 2). Hence, f1_weighted will give class 1 f1 score three times more weight than class 2's f1 score.
    a. **Error rates** will be referenced too and it is equal to (1 - score) or (1 - accuracy). We strive for low error.
2. **ROC (Receiver Operating Characteristic) Curve & AUC (Area Under the Curve) score**[13]: ROC Curve features TPR (true positive rate) on y-axis vs FPR (false positive rate) on x-axis and great to evaluable binary classifier output quality. AUC measures the two dimensional area underneath the ROC curve and is a number between 0 to 1. AUC is an aggregate measure of performance across all possible classification thresholds.

Baseline - ROC Curve for Out of Box SciKit Learn Models on Datasets[14]

To measure performance of our models, we established a baseline using default settings for all the classifiers. It's evident that the "Adult Census Dataset" is harder to solve than the "E-mail Spambase Dataset" as the AUC values are much higher and line series consistently score higher on ROC i.e. for the low FRP, it delivers a high TPR. This is because the adult census dataset has highly categorical and heterogeneous features that upon encoding, binning, etc. become a 30162 instances x 39 features dataframe in contrast to Email spam dataset (4601 instances x 58 features) which is homogeneous and continuous real numbers between 0 & 1 (4 times less data than Adult Census dataset). On comparing the AUC scores, we see that all models predict with following performance rankings on the datasets: ***Neural Net ~ AdaBoost (similar performance) > SVM > K Nearest Neighbors > Decision Tree.***

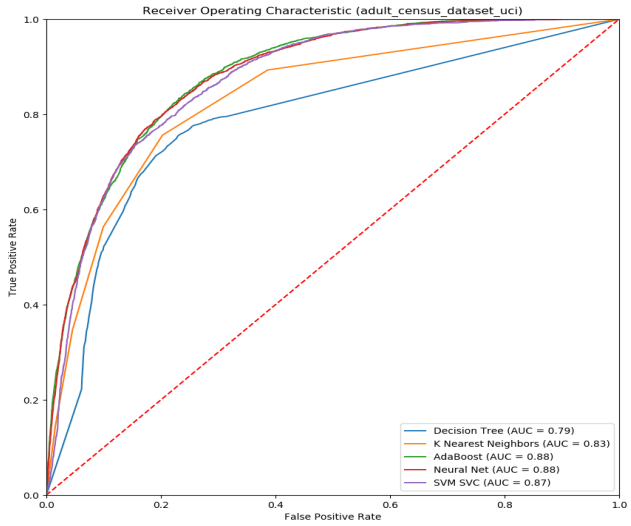


Fig 1a. ROC Curve for Baseline Models on Adult Census Data

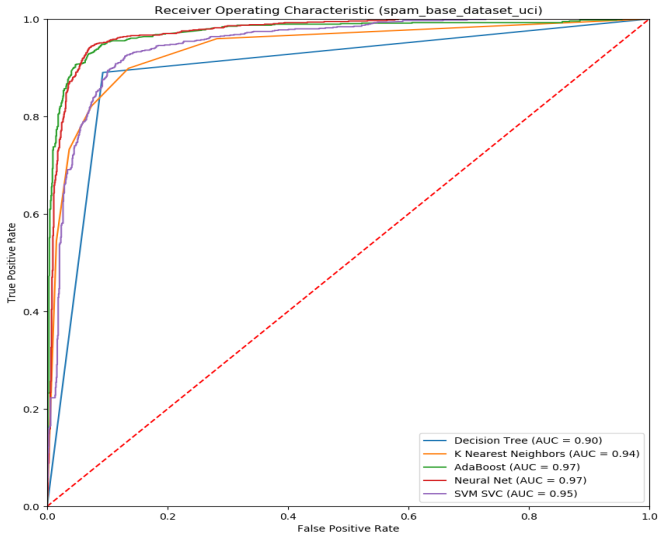
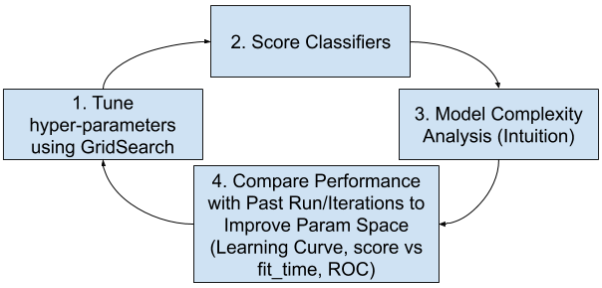


Fig 1b. ROC Curve for Baseline Models on E-mail Spam Data

Iterative Process - Bias Variance Tradeoff, Grid Search, Model Complexity & Learning Curve Analysis

`sklearn.model_selection.GridSearchCV` is then used to get an understanding of these 5 supervised learning models. For every classifier, a range of parameters is given to test and find an optimal model. These were saved using `joblib.dump`[15] and generated ROC graphs. These iterative experimental runs are available in my code submission under folders "aws_runs_*". The ROC graphs were compared with future iterations to find the best optimal model.

Approach/Idea: *Grid Search is just doing a **brute force** on the params given to it to find the best model. It is not really the most optimal because the assumption is that **Grid Search is as good as the parameter space (range of params) we give it**. We need to perform **Model Complexity analysis** to understand how each parameter & it's range really affects the model architecture, scores/errors, etc., use **Learning Curves** to understand the effect of size of training samples on performance and then build an optimal model using this empirical data from graphs. **Our goal is to reduce bias (from erroneous assumptions in classifiers) and variance (due to noise in data)**.* These params are stored in class attribute `SupervisedLearning.classifier_params_grid_search`. E.g.



Approach Taken to Implement Algorithms with the **Goal of Minimizing Bias and Variance**

```
# sklearn.tree.DecisionTreeClassifier
'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 10, 15, 20, 25, 30]
# sklearn.ensemble.AdaBoostClassifier
'n_estimators': [1, 2, 4, 8, 16, 32, 64, 100, 200]
# GridSearch called in a loop on classifiers &
classifier_params_grid_search array.
cv = StratifiedShuffleSplit(n_splits=3, test_size=0.3,
random_state=0)

grid_search = GridSearchCV(classifier, classifier_params,
cv=cv, return_train_score=True, n_jobs=n_jobs, verbose=10,
scoring=scoring_metric)
```

III. Model Complexity (MC) Analysis and Final Model Implementations

Model Complexity - Validation Curves

To find the hyperparameter ranges, model complexity analysis is performed. A dictionary of params and their ranges is built with intuition and research. For each baseline classifier and for each parameter (random_state=0 to ensure reproducibility of results), a validation curve[16] is plotted via `sklearn.model_selection.validation_curve` & `matplotlib.pyplot`[17].

```
train_scores, test_scores = validation_curve(estimator, X, y, param_name=param_name,
param_range=param_range, cv=cv, scoring=scoring, n_jobs=n_jobs, verbose=10)
```

Several parameters (3-6) were analyzed per classifier for constructing an optimal model and *many are omitted from this MC analysis due to brevity*. View source code and images for different experimental runs to see how MC analysis improved param ranges.

Decision Tree (DT)

max_depth is a pre-pruning technique (no post pruning was performed) which specifies the maximum depth a decision tree can grow to (size of the tree). Hence, ***a lower depth means the tree is underfitting*** *(see Figure 2) because training score & cross validation (CV) score on the test set are both very low i.e. error is high. This happens because the tree is unable to generalize since it doesn't grow beyond the low depth specified.* On the other hand, ***a higher depth means the tree is overfitting*** *the data and vulnerable to noise.* The training score remains high in this case but the CV score on the test set drops. We see that setting a max_depth between 10 to 15 yields the best results with the right amount of generalization.

For varying max_depth on the 2 datasets, we notice the patterns are similar. ***The decision tree makes no assumptions to modeling the different datasets and hence has low bias for max_depth 10-15.***

Some other pre-pruning params analyzed were min_samples_split, min_samples_leaf, max_leaf_nodes, min_impurity_decrease. The default criterion 'Gini' for splitting the features was used per MC analysis. ccp_alpha (post-pruning) was also analyzed using MC graph but not used for building the classifier.

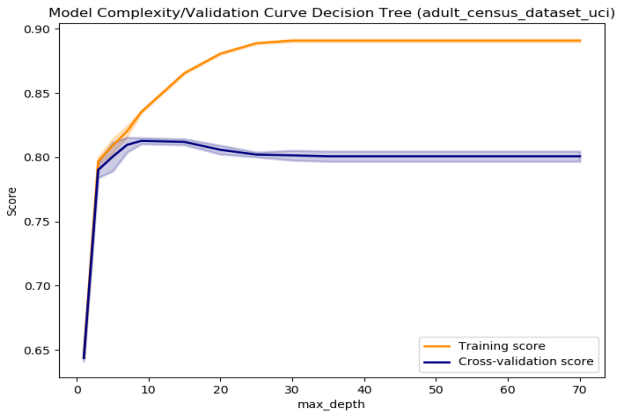


Fig 2a. MC curve of f1_score vs max_depth on Adult Census.

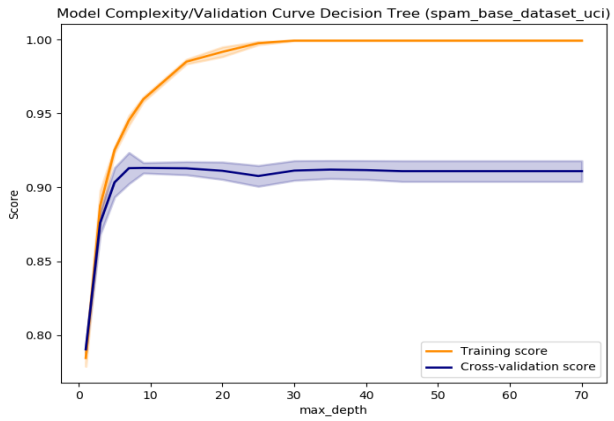


Fig 2b. MC curve of f1_score vs max_depth on Spambase.

K Nearest Neighbors

When k (number of neighbors to use for kneighbors queries) or ***n_neighbors is small, the classifier overfits*** as the training score is high but CV score on the test set is low. Again, this occurs due to excessive generalization even to noise. ***When k is large, the classifier underfits*** as training score and CV score on the test set are both lower. This happens due to low generalization to the data. On Adult Census data, an ideal k value is between 25 to 35 while on the Spambase data set the ideal k value is between 3-7. In Spambase, higher values of k lead to poor scores while lower k values lead to better scores. This means ***bias increases as k or n_neighbors increases***. Also, note that the Adult *Census dataset's ideal k value is much higher than Spambase dataset, possibly because Adult Census dataset is heterogeneous with more features and higher complexity.*

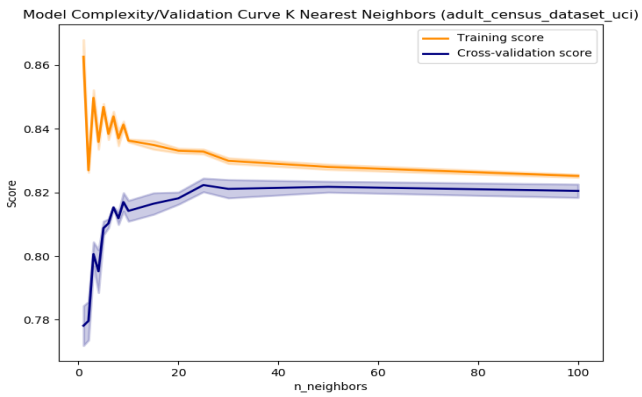


Fig 3a. MC curve of f1_score vs k on Adult Census.

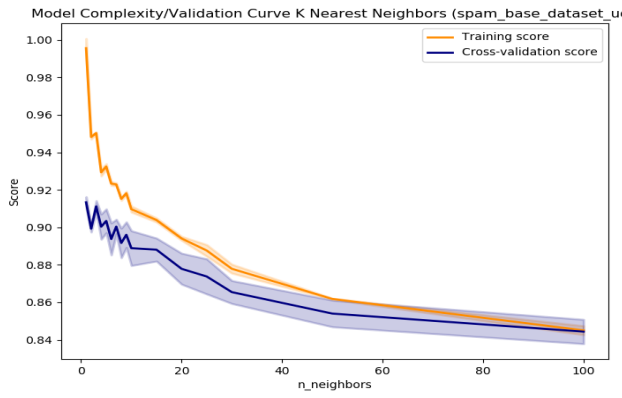


Fig 3a. MC curve of f1_scrore vs k on Spambase.

SVC Classifier (Support Vector Machine)

SVM SVC (C-Support Vector Classification) kernel functions take input data and transform it into required form. From Figure 4, it's clear that default *rbf* (radial basis function) and *linear* kernel functions give best performance (f1_score) on both datasets. *sigmoid* performs poorly on both the datasets while *poly* does well on Adult Census data but poorly on Spambase dataset. In the code experiments, *poly, linear and rbf were all used to generate MC analysis and learning curves.*

Let's select rbf due to its superior performance on both datasets for the best estimator. The kernel function is written as[18]: $(\gamma\langle x, x'\rangle + r)^d$ where γ (gamma) must be >0. *C is the Regularization Param and inversely proportional to strength of regularization (larger C implies less "generalization" or more complex estimator*. Hence, from Figure 5, ***a smaller value of C i.e.*** 10^{-3} ***leads to underfitting*** where training score and the CV score on the test set are both low (***this is also the high bias region***). On the other hand, a ***large C i.e.*** 10^{3} ***leads to overfitting*** where training score is high but CV score on test set is low. This implies ***for higher C, the bias becomes low and independent of C (also backed by conclusions from Journal paper "Bias Variance Analysis of SVMs***[19]). For Adult Census data a C=0.1 is selected & for Spambase C=100 is selected.

Gamma (not shown here) behaves similarly where a ***large γ leads to overfitting and small γ leads to underfitting***. For Adult Census data a γ=100 is selected while for Spambase γ=0.1 is selected.

For SVC with **poly kernel**, ***a large degree of polynomial means high training score but low CV score on test set as seen in Figure 6a. This leads to overfitting to data (& hence high variance).*** On the other hand, a *lower degree polynomial implies low training score and low CV score on test set i.e. underfitting the data (high bias).* Hence, for Adult Census data degree=5 leads to best generalization & for Spambase degree=1 seems ideal. In any case, due to the 'poly' kernel's lower performance in comparison to 'rbf' kernel, it's not used for finding optimal tuned SVM models.

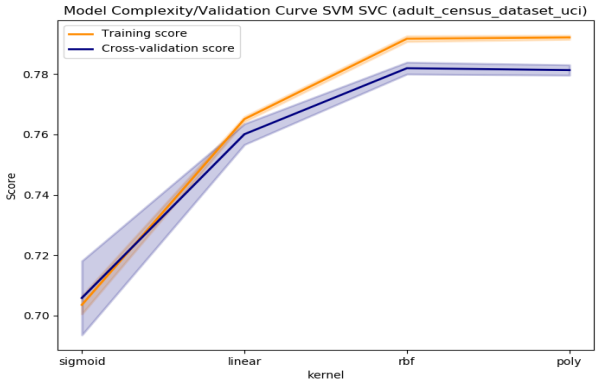


Fig 4a. MC curve of f1_score vs kernel on Adult Census.

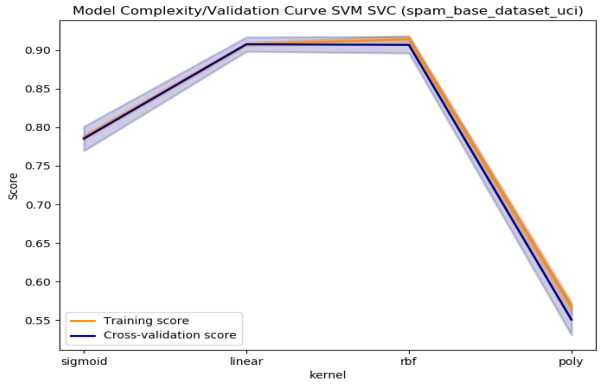


Fig 3a. MC curve of f1_score vs kernel on Spambase.

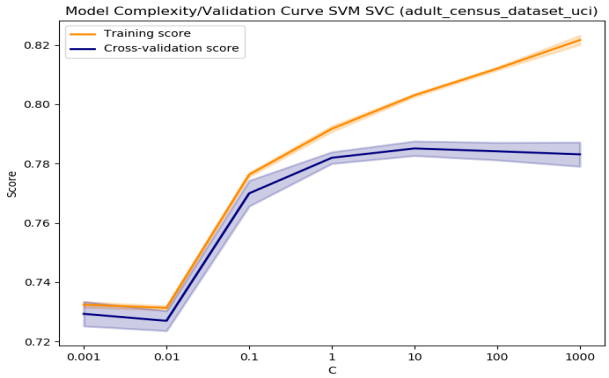


Fig 5a. MC curve of f1_score vs C on Adult Census with rbf.

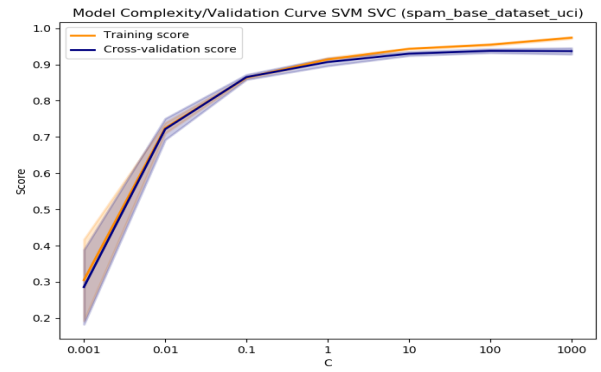


Fig 5a. MC curve of f1_score vs C on Spambase with rbf.

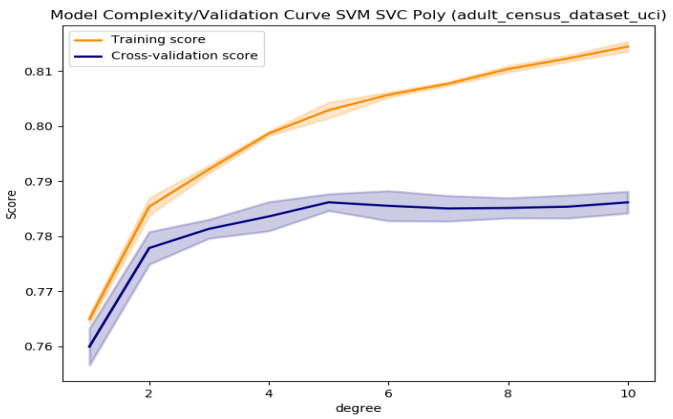


Fig 6a. MC curve of f1_score vs degree on Adult Census with poly kernel function.

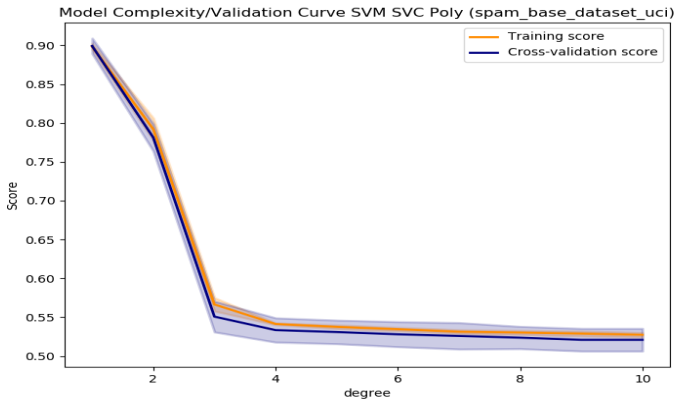


Fig 6a. MC curve of f1_score vs degree on Spambase with poly kernel function.

AdaBoost with Decision Tree Base Estimator

Adaptive Boost with hyperparameter `base_estimator=DecisionTreeClassifier(random_state=0)` of varying max_depth is used. By default, AdaBoost uses maximum n_estimators=50 i.e. up to 50 weak DecisionTree learners. Using max_depth as a pre-pruning technique again, we see that ***a higher depth means the tree is overfitting*** *the data and vulnerable to noise* (see Figure 7). The training score remains very high in this case but the CV score on the test set drops. *Since we're using boosting,* ***it's clear being aggressive about pruning yields much better results*** *(at least on the Adult Census dataset).* Still using a max_depth = 1 or 2 continues to underfit the data (as seen in DT) due to inability to generalize. We see that setting a max_depth of 3 yields the best results with the right amount of generalization on Adult Census data while setting a max_depth between 20-25 yields lowest test error or highest CV score on Spambase dataset.

MC analysis is also conducted on varying n_estimators. It's evident from Figure 8 for both datasets that ***low number of weak learners leads to underfitting*** *of data and hence lower CV score on test set and training score.* This is due to inability to generalize. A large number of weak estimators may lead to overfitting (as seen in Fig 8b.) where CV score on test set is low but training score is high. *In general, for AdaBoost a high number of estimators seems a lot more performant due to the power of adaptively combining weak learners to learn from hard to solve data points iteratively* i.e. n_estimators>=40 for Adult Census dataset or n_estimators in {100, 150} for Spambase dataset.

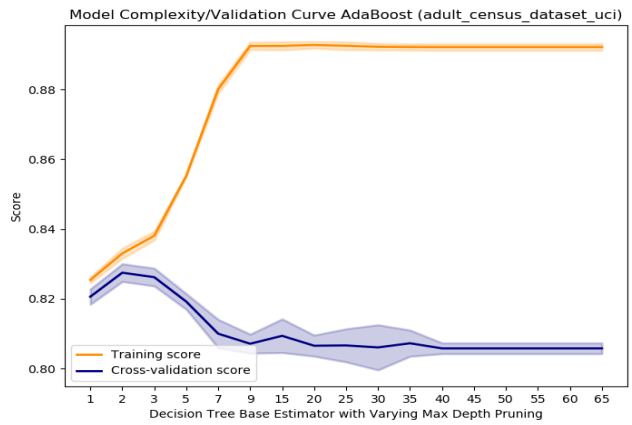


Fig 7a. MC curve of f1_score vs max_depth on Adult Census.

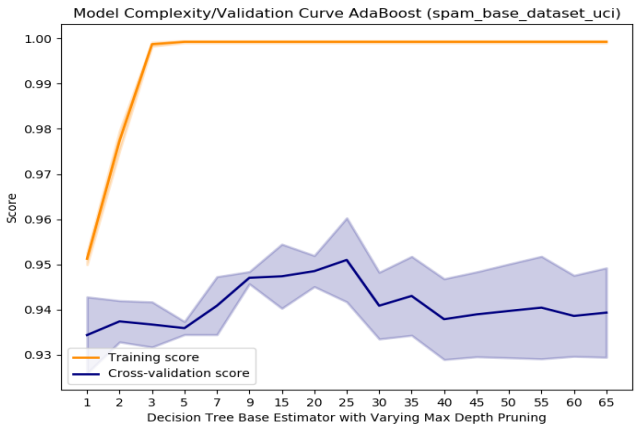


Fig 7b. MC curve of f1_score vs max_depth on Spambase.

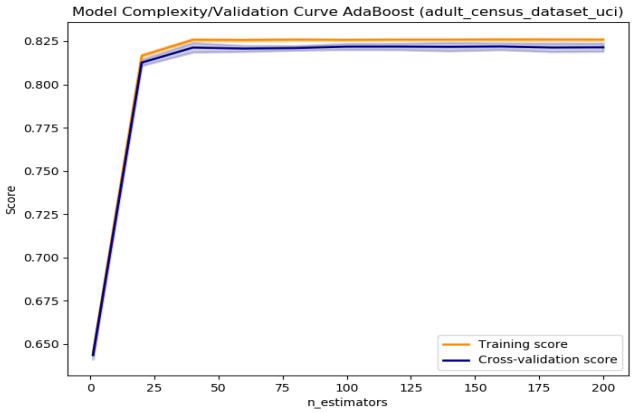


Fig 8a. MC curve of f1_score vs n_estimators on Adult Census.

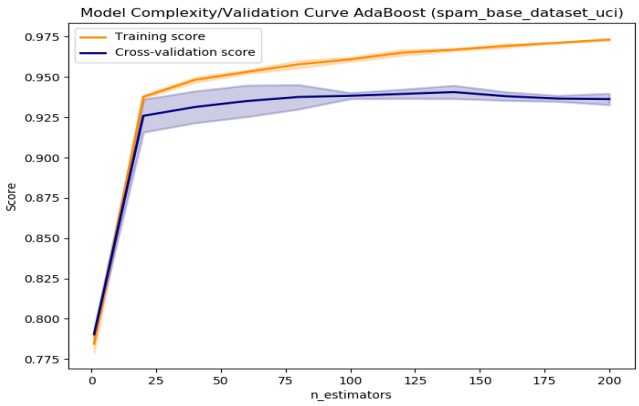


Fig 8b. MC curve of f1_score vs n_estimators on Spambase.

MLP Classifier (Neural Net)

For Neural Net, a Multi-layer Perceptron (MLP) classifier is used with varying the parameters hidden_layer_sizes. It's clear that increasing the number of neurons or layers doesn't necessarily improve performance. Per Figure 9, low number of neurons per layer or lesser layers leads to low training score and low CV score i.e. the estimator is underfitting. For a ***high number of neurons or higher number of layers, the training score is high and the validation score is low i.e. the estimator is overfitting***. For Adult Census data hidden_layer_sizes=(8,) (i.e. 8 neurons) & for Spambase hidden_layer_sizes=(256,) is selected. MC analysis is also performed on params *alpha, max_iter, activation functions, learning_rate, max_iter* (omitted due to brevity). ***max_iter is interesting because reducing the number of max_iter performs better (higher CV score)*** on Adult Census dataset while increasing max_iter performs better (higher CV score on test set) on Spambase dataset.

For param *activation functions* (not shown due to brevity), ***tanh & relu have the best performance*** *with high CV score on test data and high training score.* Hence, we'll be using the default 'relu' activation function for our optimal model.

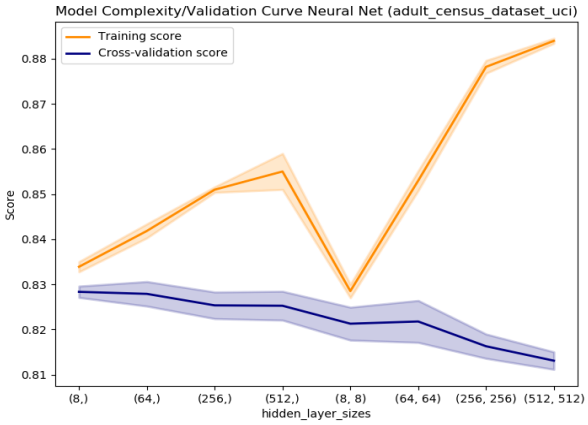


Fig 9a. MC curve of f1_score vs hidden_layer_sizes on Adult Census.

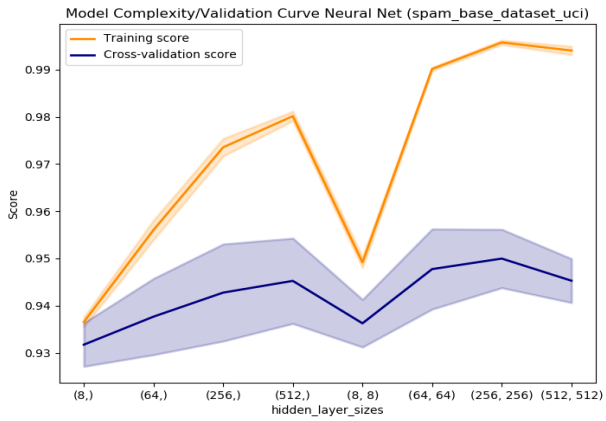


Fig 9b. MC curve of f1_score vs hidden_layer_sizes on Spambase.

Learning Curves

To understand how each classifier behaves on varying amounts of data, learning curves are plotted. The best models found so far are used based on the ideal params discovered via iterative MC analysis & Grid Search tuning process.
`sklearn.model_selection.learning_curve`[20] & `matplotlib.pyplot`[17] are used on these models with ShuffleSplit CV.

```
cv = ShuffleSplit(n_splits=3, test_size=0.2, random_state=0)
scoring = 'f1_weighted'
train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(best_estimator, X, y,
cv=cv, scoring=scoring, n_jobs=n_jobs, train_sizes=train_sizes, return_times=True, verbose=10)
```

Decision Tree

When the number of training examples is small, *training score is high (as seen in Figure 10) while CV score on test set is low*. As the number of training examples increases, the training score and CV score seemingly come closer (but don't converge still). The ***major difference between training score and CV score on the test set is an indication that DTs are susceptible to high variance*** in data. *This is* ***more apparent on Spambase dataset which is real continuous data*** *vs Adult census' categorical data*. This likely happens due to overfitting to the noise in the data.
We saw above in MC analysis that by ***using pruning techniques, we were able to reduce overfitting to the data*** and it's clear that decreasing model complexity by controlling max_depth parameter will help reduce variance.

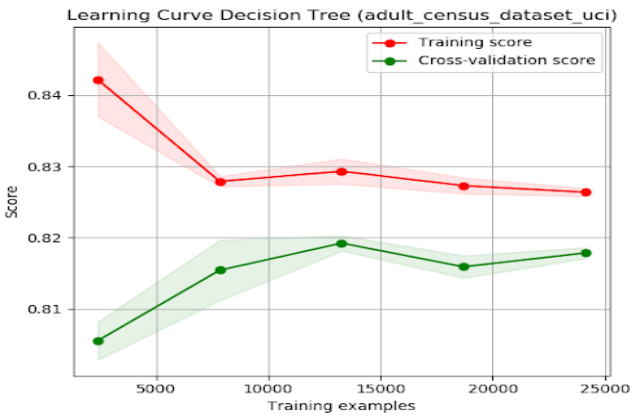


Fig 10a. f1_score vs training examples on Adult Census.

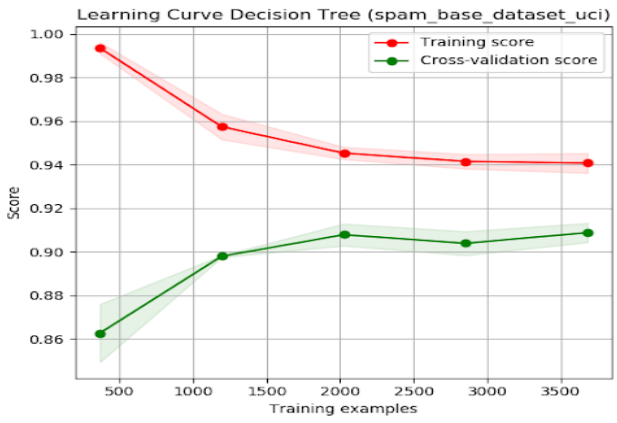


Fig 10b. f1_score vs training examples on Spambase.

K Nearest Neighbors

On the *Spambase dataset, it's evident that for low training examples size, the difference between CV score on test set and training score is very high initially (Figure 11b).* This is *likely because of overfitting* to the (noise in) dataset & suffers from very high variance. As training examples increase, the difference becomes smaller but still has high variance. On the Adult Census data,

we still see a higher training score than CV score on the test set but the difference is much less compared to Spambase. Moreover, as training examples size increases CV score increases.

This is likely because *n_neighbors selected by the process above is 30 for adult census dataset and 1 for Spambase initially*. ***Hence, when k is small & training examples size is small classifier has high variance due to overfitting. To improve the performance, k= 4 is set for Spambase classifier. This leads to much better performance as seen in Figure 11c!***

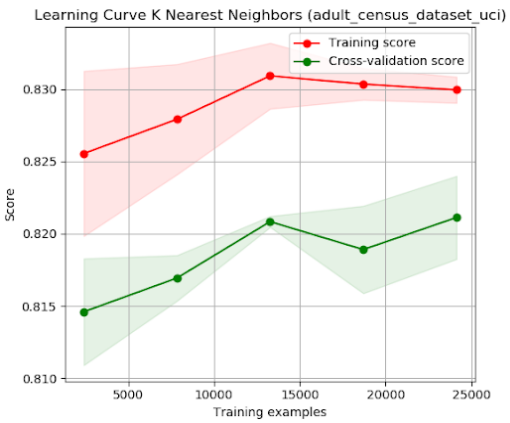


Fig 11a. f1_score vs training examples on Adult Census.

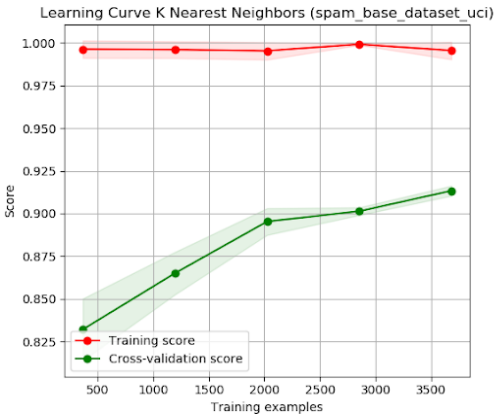


Fig 11b. f1_score vs training examples on Spambase before improvement (k=1).

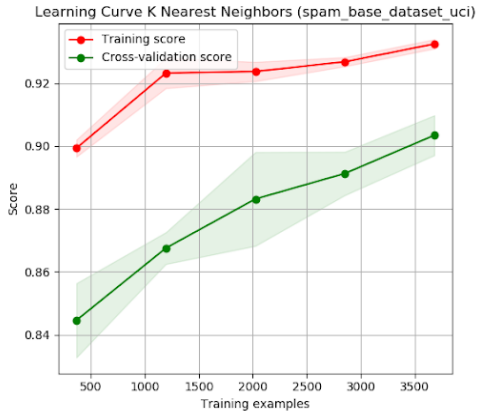


Fig 11c. f1_score vs training examples on Spambase after k increase to 4.

SVC Classifier (Support Vector Machine)

Radial Basis Function Kernel

On Spambase data, we see ***as training examples size increases, CV score on test set increases and training score slightly decreases*** *(nearly converges)*. Though, for lower size of training examples, the training score is higher but CV score on the test set is lower, implying overfitting to the smaller datasets and suffers from high variance.

On Adult census data, we see similar training score and CV score that's increasing with training examples of size up to 14000 samples but both CV score and training score drop rapidly for training examples of greater sizes. This happens because SVMs are working with highly dimensional data (large number of features) and ~30,000 x 39 size dataset. Hence, they hit a max_iter=10,000 limit set before these classifiers can converge. This limit is set to ensure some SVC classifier fits within reasonable time. As scikit docs say[21], "The fit time scales at least quadratically with the number of samples and ***may be impractical beyond tens of thousands of samples***." We'll analyze this below in greater detail when looking at wall clock time.

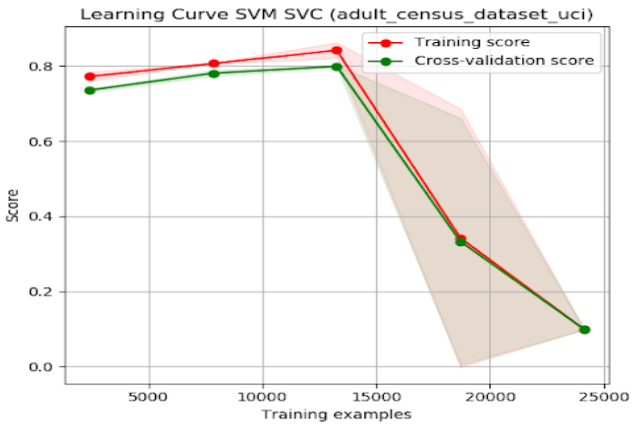


Fig 12a. f1_score vs training examples on Adult Census with rbf.

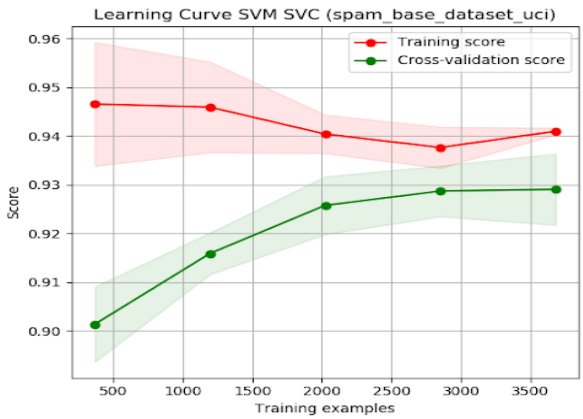


Fig 12b. f1_score vs training examples on Spambase with rbf.

Poly Kernel Function

On Spambase, *CV score on test set & training score increases as training examples size increases*. This shows increasing performance with increasing training size. Maybe collecting more data may continue to increase performance here with the caveat of scaling limits discussed above.

On Adult Census data, ***CV score on test set remains flat but training score slightly decreases*** *as training examples size increases*. Though, for lower size of training examples, the training score is high but CV score on the test set is low, implying some overfitting to the smaller datasets (& their noise).

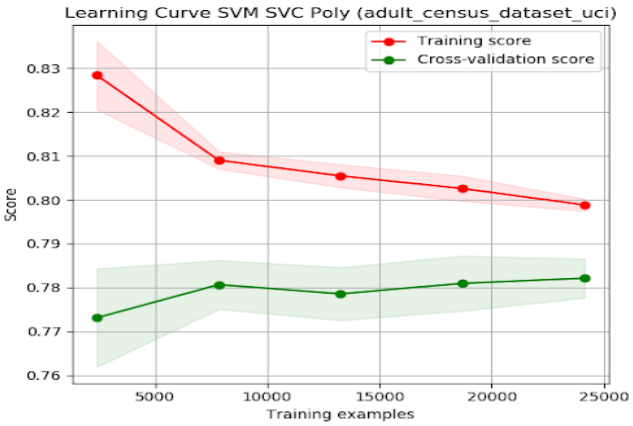


Fig 13a. f1_score vs training examples on Adult Census with poly.

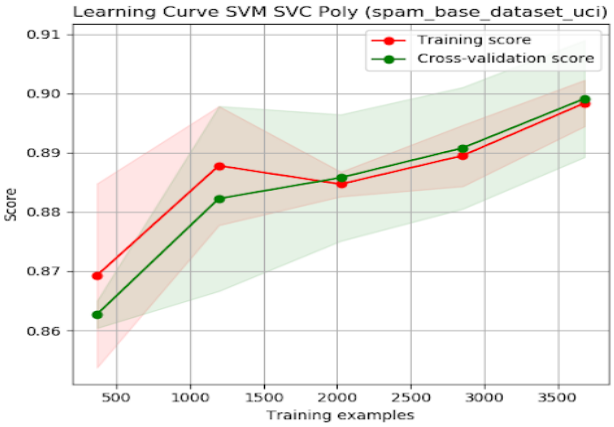


Fig 13b. f1_score vs training examples on Spambase with poly.

AdaBoost with Decision Tree Base Estimator

AdaBoost ensemble and boosting classifier is well known to avoid overfitting to datasets because of it's model architecture consisting of several weak learners learning from "hard" to get right/predict instances iteratively. This is evident from Figure 14a on Adult Census dataset, *where the increase in training examples size reduces the training score and increases the CV score on test data*. Though, when the training examples size is small, the training score is high & CV score on test set is low. This implies overfitting to the data and after 15000 examples both lines become flat.

On Spambase data, ***the training score consistently remains extremely high but CV score on the test set is lower*** but still rising as the number of training examples increases. This means that AdaBoost is overfitting to the data and we may be able to improve this further through regularization or further pruning.

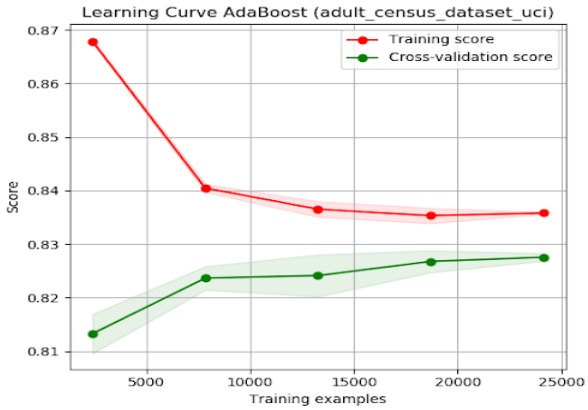


Fig 14a. f1_score vs training examples on Adult Census.

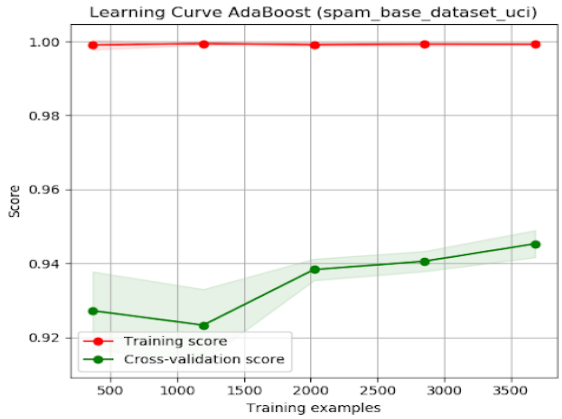


Fig 14b. f1_score vs training examples on Spambase.

MLP Classifier (Neural Net)

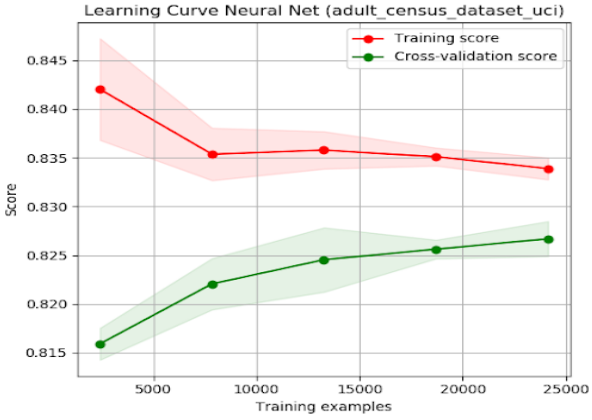


Fig 15a. f1_score vs training examples on Adult Census.

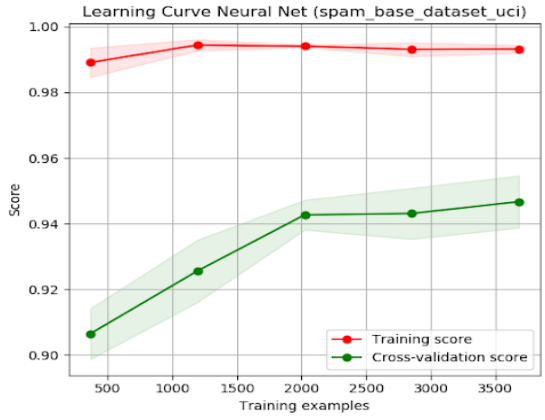


Fig 15b. f1_score vs training examples on Spambase.

MLPs seem to show similar trends as AdaBoost (may be because they both seem very equally powerful in terms of performance on both datasets). On the *Adult Census dataset, as the training examples size increases, the CV score on the test set increases and training score decreases.* When the training examples size is small, the difference between training score and CV score is high. *This implies overfitting to the dataset* and after 15000 examples both lines become flat.

On the Spambase dataset, ***the training score consistently remains extremely high but CV score on the test set is lower*** but still rising as the number of training examples increases. Though, it's error becomes as low as 0.05 (1 - score) with larger training examples size.

IV. Further Analysis of Results - Comparison of Supervised Learning Models

Wall Clock Time Comparison

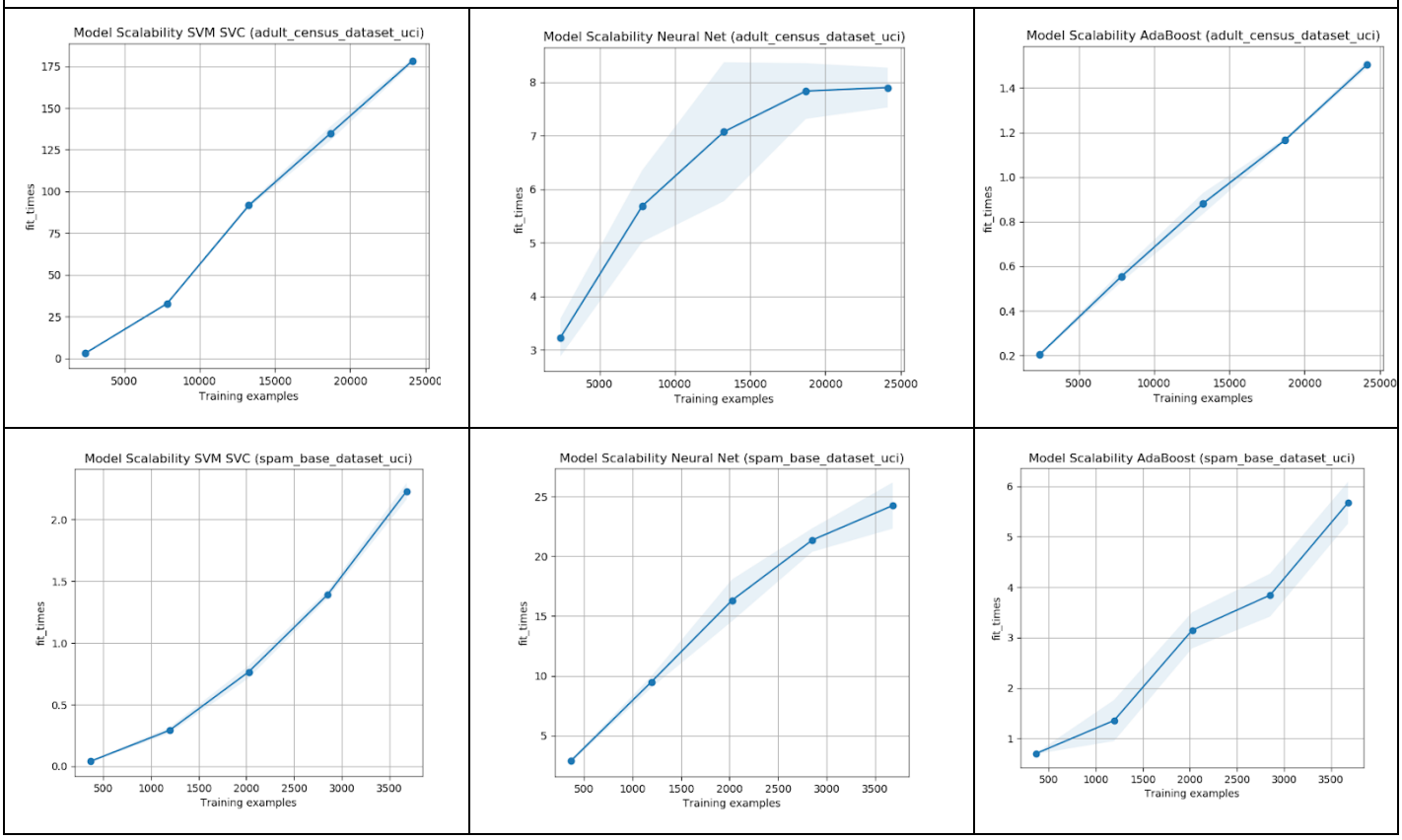
Model Fit Times Scalability

Fit Times Performance Summary: Decision Trees (Best) > KNN > SVM SVC For Low Dimensional Homogeneous Data > AdaBoost > Neural Net > SVM SVC For High Dimensional Categorical Heterogeneous Data (Worst)

From Figure 16, it's clear that ***generally speaking all classifiers' fit times (in seconds) increase linearly as training examples increase***. In some cases we see either slightly exponential growth (SVM SVC rbf on Spambase or K Nearest Neighbors on Adult Census) or logarithmic growth (Neural Net fit_times on Adult Census data).

Unsurprisingly, SVM SVC rbf takes the longest to perform fit on Adult Census dataset due to high dimensionality (around 175 seconds for 25000 training examples). For SVM SVC, some features with lower correlation to target values were purposefully removed to speed up the computation such as race & occupation. This reduced the # of features from 39 to 20! In contrast, SVM SVC rbf performed much faster on Spambases homogeneous real contiguous data as it took merely 2 seconds to fit on its 3600 training examples. Some DT & KNN graphs were omitted due to brevity.

Fig 16. fit_times (in seconds) vs number of training examples on datasets. DTs & KNN are omitted due to brevity.



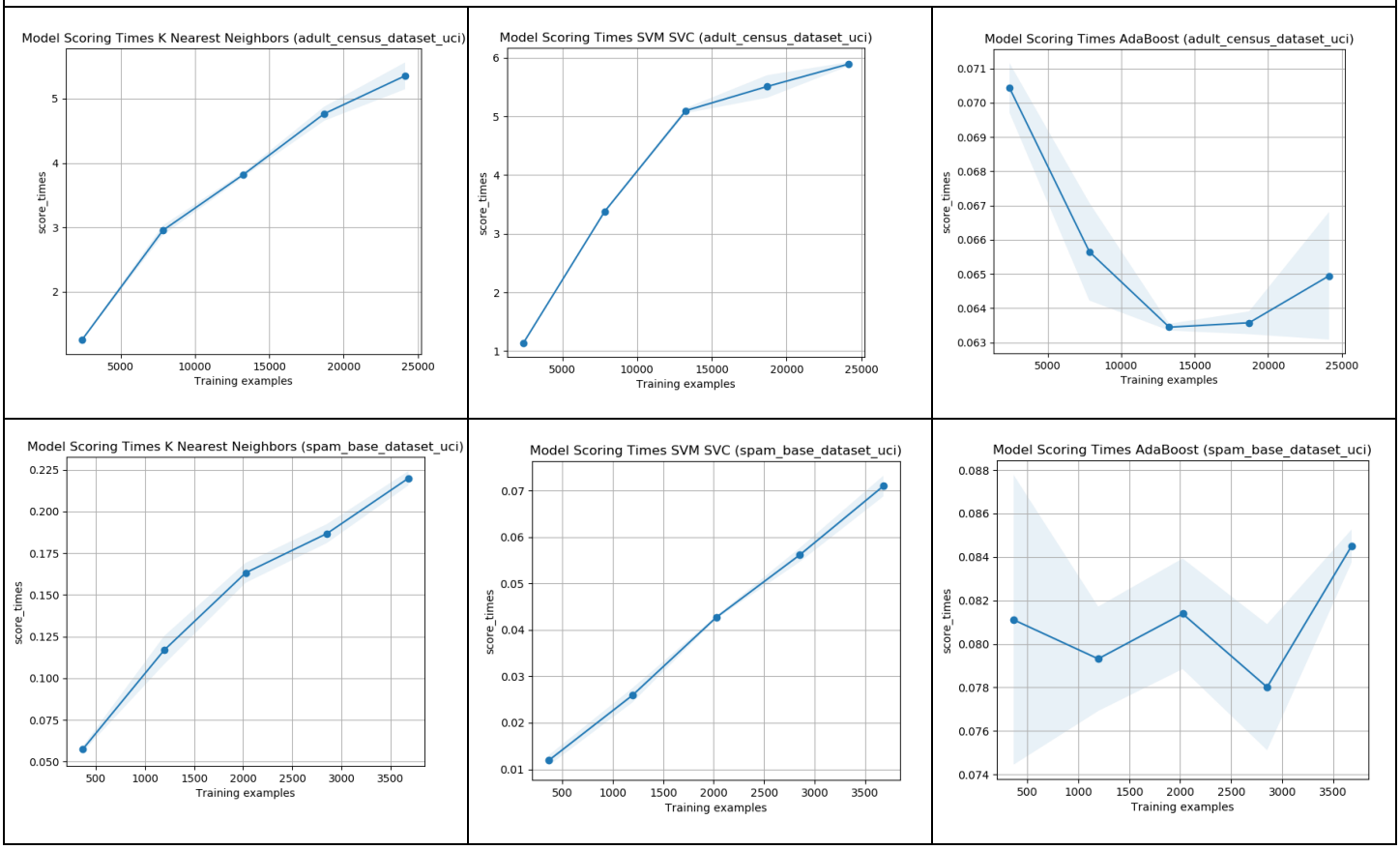
Model Scoring/Predict Times Scalability

Scoring Times Performance Summary: KNN > SVM SVC rbf > AdaBoost > Neural Net > Decision Trees

Unsurprisingly, **KNN takes the longest to score/predict as expected because KNN is a "lazy learner"** as it memorizes the training dataset and doesn't learn a discriminative function. SVM SVC rbf also takes a long time to score on Adult Census dataset likely due to similar reasons discussed before (high complexity of model & high dimensionality). *Both KNN and SVC*

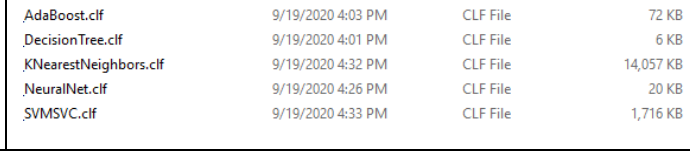
score_times grow linearly with increasing training example size. MLP Neural Net & DTs (omitted) remain flat in terms of score_times vs training example size growth. Surprisingly, AdaBoost score_times decreases as training examples size increases!

Fig 17. score_times (in secs) vs number of training examples on datasets. DTs & MLP Neural Net are omitted due to brevity.



A Short Note on Disk Space Consumption of Classifier Dumps: **KNN > SVM SVC rbf > AdaBoost > Neural Net > Decision Trees**

As seen in the image (for Adult Census data), you'll notice that ***KNN classifier occupies most disk on file followed by SVM SVC. This isn't surprising because as noted before KNN is a lazy learner that "memorizes"*** the training set.



AdaBoost.clf	9/19/2020 4:03 PM	CLF File	72 KB
DecisionTree.clf	9/19/2020 4:01 PM	CLF File	6 KB
KNearestNeighbors.clf	9/19/2020 4:32 PM	CLF File	14,057 KB
NeuralNet.clf	9/19/2020 4:26 PM	CLF File	20 KB
SVMSVC.clf	9/19/2020 4:33 PM	CLF File	1,716 KB

Final Models Implemented

After performing the iterative process discussed earlier, below are final model implementations that yield best performance.

Fig 18a. Best classifiers on Adult Census dataset.

```
DecisionTreeClassifier(max_depth=11,max_leaf_nodes=30
,min_samples_leaf=0.001,random_state=0)

KNeighborsClassifier(n_neighbors=30)

AdaBoostClassifier(base_estimator=DecisionTreeClassif
ier(max_depth=3,random_state=0),learning_rate=0.3,ran
dom_state=0)

MLPClassifier(alpha=1e-06,hidden_layer_sizes=(8,),max
_iter=10000,random_state=0,warm_start=True)

SVC(C=0.1,cache_size=5000,class_weight='balanced',gam
ma=100,max_iter=10000,probability=True,
random_state=0)
```

Fig 18b. Best classifiers on Spambase dataset.

```
DecisionTreeClassifier(max_depth=11,max_leaf_nodes=3
0,min_samples_leaf=0.001,random_state=0)

KNeighborsClassifier(n_neighbors=4)

AdaBoostClassifier(base_estimator=DecisionTreeClassi
fier(max_depth=5,random_state=0),learning_rate=1,
n_estimators=200,random_state=0)

MLPClassifier(alpha=0,hidden_layer_sizes=(256,),max_
iter=10000,random_state=0,warm_start=True)

SVC(C=100,cache_size=5000,class_weight='balanced',ga
mma=0.1,max_iter=10000,probability=True,random_state
=0)
```

V. Conclusion

Upon performing the iterative process of model complexity analysis, learning curve analysis and grid search, we've optimized our models to have lower bias and variance wherever possible. It's evident from figure 19 below that classifiers such as DT & KNN that initially gave poor performance significantly improved. AdaBoost (with an ensemble of weak learners & boosting) & Multi-layer Perceptron (Neural Networks) are still the best algorithms/classifiers for prediction on both the datasets. ***Moreover, for all classifiers, the lines moved/bent towards top left with higher TRR & low FPR showcasing a major improvement.***

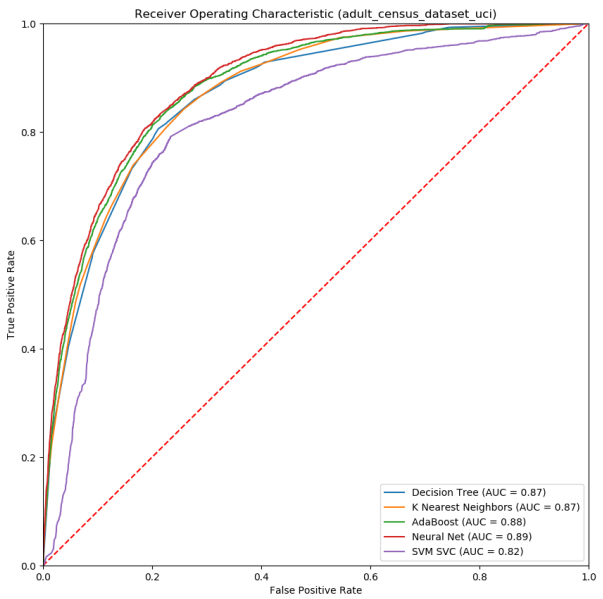


Figure 19a. ROC Curve for best models on Adult Census Data

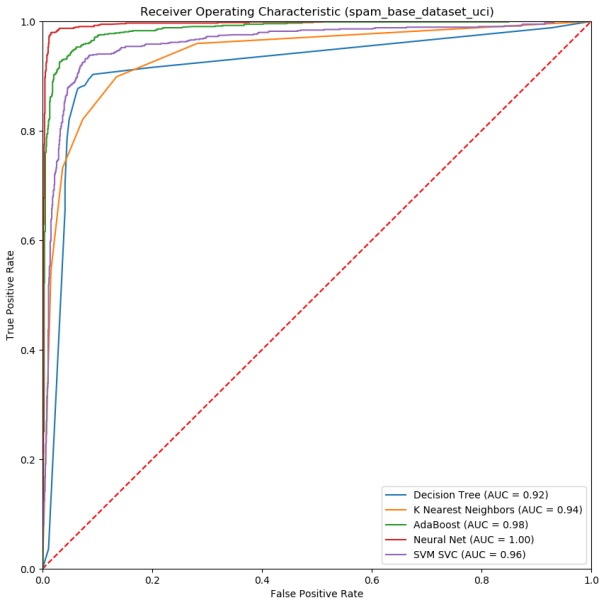


Figure 19b. ROC Curve for best models on Spambase Data

Decision Tree AUC: increased by 0.08
K Nearest Neighbors AUC: increased by 0.04
SVM SVC using RBF AUC: didn't improve
AdaBoost AUC: remained same
MLP Classifier (Neural Net) AUC: improved by 0.01

Decision Tree AUC: increased by 0.02
K Nearest Neighbors AUC: remained same
SVM SVC using RBF AUC: increased by 0.01
AdaBoost AUC: increased by 0.01
MLP Classifier (Neural Net) AUC: increased by 0.03

VI. Citations

[1] Sarah Guido, Andreas C. Müller, "Introduction to Machine Learning with Python", Published by O'Reilly Media. [E-Book]
[2] Ronny Kohavi & Barry Becker, Data Mining & Visualization, Silicon Graphics, "Census Income Data Set". [URI]
[3] Mark Hopkins, Erik Reeber, George Forman, Jaap Suermondt, Hewlett-Packard Labs, "Spambase Data Set". [URI]
[4] Justin M. Rao, Microsoft Research, David H. Reiley, Google, Inc. "The Economics of Spam" [URI]
[5] Center for Information Security, "Fall 2019 Threat of the Quarter: Ryuk Ransomware". [URI]
[6] Aleksey Bilogur, Boyan Angelov, "Missingno: Flexible & easy-to-use missing data visualizations". [URI]
[7] Chris Moffitt, "Binning Data with Pandas qcut and cut". [URI]
[8] PyData Development Team, "Pandas". [URI]
[9] Swetha Lakshmanan, "Standardize/Rescale Your Data?". [URI]
[10] Zixuan Zhang, "Understand Data Normalization in ML". [URI]
[11] Robert DeFilippi, "Standardize or Normalize? — Examples in Python". [URI]
[12] Scikit Learn, "Metrics and scoring: quantifying the quality of predictions". [URI]
[13] Google, "Classification: ROC Curve and AUC". [URI]
[14] Scikit Learn, "Receiver Operating Characteristic (ROC)". [URI]
[15] Joblib, "Persist Python Object to File". [URI]
[16] Scikit Learn, "Plotting Validation Curve". [URI]
[17] Matplotlib, "Visualization with Python". [URI]
[18] Scikit Learn, "Support Vector Machines: Kernel Functions". [URI]
[19] Giorgio Valentini & Thomas G. Dietterich, "Bias-Variance Analysis of Support Vector Machines for the Development of SVM-Based Ensemble Methods". [E-Journal]
[20] Scikit Learn, "Plotting Learning Curve". [URI]
[21] Scikit Learn, "sklearn.svm.svc". [URI]