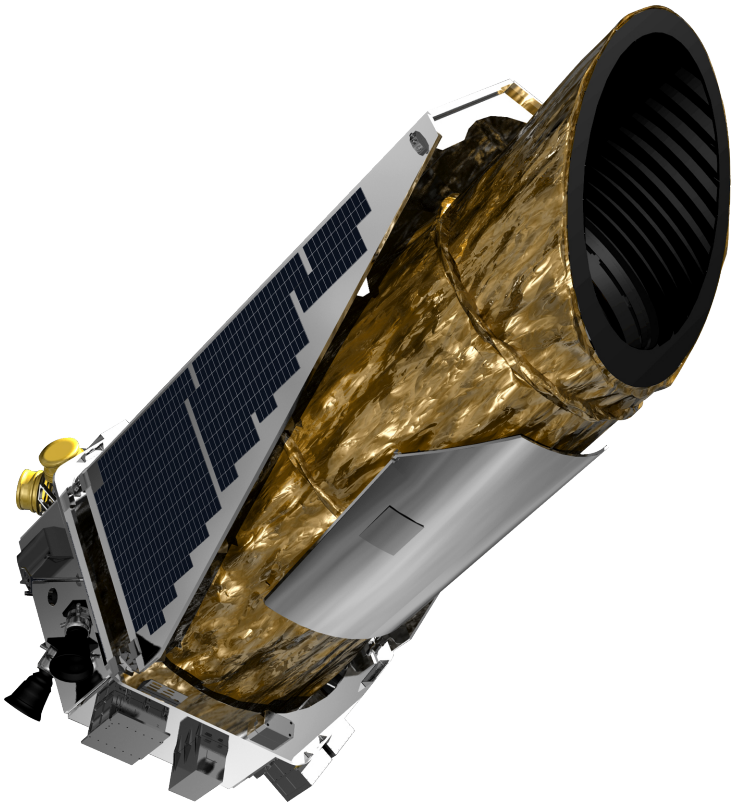
NASA Kepler Exoplanet Search

Data Mining

December 23, 2020



Aleix Boné
Eduard Bosch
David Gili
Albert Mercadé

Contents

List of Figures

List of Tables

1 Introduction

Space exploration is one of the great challenges humankind faces. Not only stemming from our innate curiosity and eagerness to always know more, but also from the ever-growing need to find other planets where our species can live. After all, the human species is increasingly endangered due to climate change, among other reasons, and while we might be able to overcome this dangers and the threats might seem very small today, the risk of Earth becoming uninhabitable someday is real.

To pursue this goal, humans have built all kinds of telescopes that work from land, such as National Radio Astronomy Observatory (NRAO) or the European Northern and Southern Observatories, and from space, including the Hubble Space Telescope. One of such telescopes is the Kepler Space Observatory. It was operational between May 2009 and October 2018, when it ran out of fuel. During its time in service, it helped find over 2,600 exoplanets, which account for around 60% of all known exoplanets.

Data about all the potential exoplanets recorded by Kepler began to be published in 2010. The dataset we used was published by NASA on Kaggle in October 2017.

Our goal for this project was to build a prediction model that could correctly classify candidate exoplanets as Confirmed or False Positives. To do so, and to find the best possible model, we pre-processed the data and fed it to four different machine learning algorithms: Naive Bayes, K-NN, Decision Trees and Support Vector Machines. We then compared the outcome of these algorithms to find which works best for our dataset and target, and aimed to find the reasons why the algorithms behaved the way they did.

2 Description of the original data

2.1 The dataset

This dataset gathers close to 10,000 observations made by NASA's Kepler Space Observatory in pursuit of finding new exoplanets that could potentially be habitable. We obtained the dataset from Kaggle[1], the online data science and machine learning community.

9.564 observations were made by the Kepler observatory between 2009 and 2017, when this dataset was published, and all are contained in the dataset. Moreover, it is comprised of 50 variables, most of them numerical and a few categorical and boolean.

Regarding missing data, overall there are 40.557 values missing, or what is the same 8.48% of the dataset. However, if we take out the three variables that are largely missing: `kepler_name` (76.01% missing), `koi_teq_err1`(100% missing) and `koi_teq_err2`(100% missing), the missing data drops to just under 3.15%. Out of the remaining variables, most have a missing data rate of around 3-4% with the exception of `koi_score`, which stands shy of 16%.

2.2 Target Column

In this dataset, the target of classification is quite clear: `koi_disposition`. This variable classifies the exoplanets into four different categories:

CANDIDATE Observations that are still awaiting confirmation.

FALSE POSITIVE Observations that ended up not being Exoplanets.

CONFIRMED Exoplanet observations that have been confirmed.

NOT DISPOSITIONED There are no entries with this variable since the Kepler satellite was decommissioned in November 2019, and all the data has already been classified in the other categories.

We won't consider CANDIDATE exoplanets, as we want to train and test our models with examples we know for sure are (CONFIRMED) or aren't (FALSE POSITIVE) exoplanets.

[1]Kepler Exoplanet Search Results dataset on Kaggle: https://www.kaggle.com/nasa/kepler-exoplanet-search-results

2.3 Column Description

In Table 1, we look at the dataset's variables: what they mean, their type and what values they can take.

Table 1: Column description [§]

Variable	Type[†]	Description	Values
rowid	N	Row number	[1-9564]
kepid	N	Target identification number, as listed in the Kepler Input Catalog	
kepoi_name	N	A number used to identify and track a Kepler Object of Interest (KOI)	
kepler_name	N	Kepler number name identifying the planet	
koi_disposition	CN	Category of the KOI from the Exoplanet Archive	CANDIDATE FALSE POSITIVE CONFIRMED
koi_score	N	A value between 0 and 1 that indicates the confidence in the KOI disposition	[0.0-1.0]
koi_pdisposition	CN	Pipeline flag that designates the most probable physical explanation of the KOI	CANDIDATE FALSE POSITIVE
koi_fpflag_nt	CN	Flag indicating the KOI has a light curve that is not consistent with that of a transiting planet	0 1
koi_fpflag_ss	CN	Flag indicating the KOI is observed to have a significant secondary event, transit shape, or out-of-eclipse variability	0 1
koi_fpflag_co	CN	Flag indicating that the source of the signal is from a nearby star	0 1
koi_fpflag_ec	CN	Flag indicating the KOI shares the same period and epoch as another KOI	0 1
koi_period[‡]	N	The interval between consecutive planetary transits	
koi_time0bk[‡]	N	The time corresponding to the center of the first detected transit in Barycentric Julian Day (BJD) minus a constant offset of 2,454,833.0 days	

Variable	Type†	Description	Values
koi_impact‡	N	The time corresponding to the center of the first detected transit in Barycentric Julian Day (BJD) minus a constant offset of 2,454,833.0 days	
koi_duration‡	N	The duration of the observed transits	
koi_depth‡	N	The fraction of stellar flux lost at the minimum of the planetary transit	
koi_prad‡	N	The radius of the planet	
koi_teq	N	Approximation for the temperature of the planet	
koi_insol‡	N	Insolation flux i.e. another way to give the equilibrium temperature	
koi_model_snr	N	Transit depth normalized by the mean uncertainty in the flux during the transits	
koi_tce_plnt_num	CN	TCE Planet Number federated to the KOI	[1-8]
koi_tce_delivname	CN	TCE delivery name corresponding to the TCE data federated to the KOI	q1_q17_dr25_tce q1_q16_tce q1_q17_dr24_tce
koi_steff‡	N	The photospheric temperature of the star	
koi_slogg‡	N	The base-10 logarithm of the acceleration due to gravity at the surface of the star	
koi_srad‡	N	The photospheric radius of the star	

† N = Numerical, CN = Categorical Nominal

‡ These variables have two corresponding columns named koi_VARNAME_err1 and koi_VARNAME_err2 that represent the +/− error respectively. We didn't include them in the table as they are repetitive and made it more confusing.

§ Descriptions obtained from NASA Kepler exoplanet KOI documentation [3]

3 Description of pre-processing of data

3.1 Feature Removal

Initially, we removed all of the variables which didn't provide any useful information such as names and identification numbers: `kepid`, `rowid`, `kepoi_name`, `kepler_name` or `koi_tce_delivname`.

Next, we investigated the presence of missing data in the dataset, and we found that three variables were mostly missing. Specially `koi_teq_err1` and `koi_teq_err2`, which represent the positive and negative error of `koi_teq` respectively, are completely missing. Therefore, we proceeded to remove them as they didn't provide any information whatsoever. How we treated the rest of the missing values will be explained in an upcoming section.

We also removed four features that were flags representing whether the exoplanet measurements had passed some tests. Since this flags directly impact the target feature, as `koi_disposition` will be FALSE NEGATIVE when it fails at least one of the tests and CONFIRMED when it passes all the tests and is confirmed, and are calculated using the raw data thus simplifying the problem. Furthermore, the main objective of this classification problem is to work directly with the raw Kepler measurements and not with processed data such as these flags.

Furthermore, there are two other columns, `koi_pdisposition` and `koi_score`, that we decided to erase as well, as `koi_pdisposition` represents the predicted `koi_disposition` (our target) and `koi_score` illustrates the level of confidence of that prediction, based on the aforementioned flags. Hence, they are directly linked to our target as well. As a matter of fact, in close to 70% of the 7316 remaining examples the `koi_disposition` and `koi_pdisposition` have the same value.

3.2 Example Removal

As explained in the description of the original dataset, we decided to remove all examples that had `koi_dispoistion` equal to CANDIDATE, as we want to train and validate our models using exoplanets that we know for sure whether they are or aren't confirmed, and not with examples that are pending for validation. This resulted in the elimination of 2248 entries out of the original 9564, leaving us with 7316 examples.

3.3 Value Transformation

Because some algorithms only work with numerical features, we had to transform our target column from categorical to numerical. Since we had removed all instances of CANDIDATE, there were no examples of NOT DISPOSITIONED and the categories don't follow a particular order, we opted for One-Hot encoding. Furthermore, as we only had two categories, we didn't need to create extra columns, we simply transformed the existing `koi_disposition` column to be 1 when the example is CONFIRMED and 0 when it isn't, meaning it is a FALSE POSITIVE.

3.4 Error Treatment

We scanned our dataset looking for errors and we weren't able to find any. We checked that numerical values were within the plausible ranges, like for example that the planet radius or temperatures were positive. We also verified that all categorical features only contained the expected categories.

3.5 Missing Data Treatment

For the remaining missing data, we decided to fill those empty cells with the mean of the feature. We were able to do that because all of our columns with missing values were of numerical type.

3.6 Feature Selection

The next step in our pre-processing script, consists in reducing the number of features by selecting the ones that provide the best results. To do that, we first need to standardize the data, by transforming all variables so they have a mean equal to 0 and standard deviation to 1.

Then, we loop through all possible number of features to be selected, excluding the target, that is from 0 to 36, the number of remaining features, and do cross validation using Knn, saving the average score for each number of selected features. Figure 1 below, shows the results that we obtained. As we can see, we obtained the best score when only using 25 out of the 36 features.

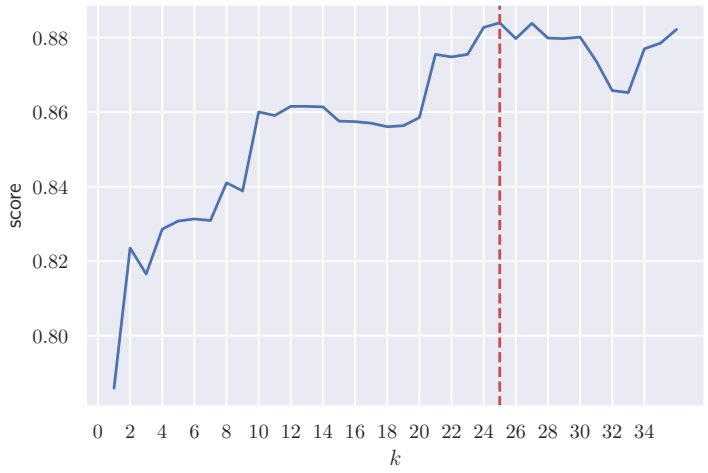


Figure 1: Cross validation score for different k values

Once we knew that using the best 25 variables provided the best scores, we selected those 25 columns, which we can see below in table 2.

Table 2: Selected features (25)

	Variable	Score		Variable	Score
1	koi_steff_err1	0.193758	14	koi_slogg_err1	0.120881
2	koi_prad	0.184690	15	koi_steff	0.115671
3	koi_steff_err2	0.170011	16	koi_period_err2	0.109981
4	koi_prad_err1	0.164256	17	koi_period_err1	0.108709
5	koi_duration_err2	0.156999	18	koi_insol_err2	0.108357
6	koi_prad_err2	0.154296	19	koi_depth	0.106608
7	koi_duration_err1	0.153266	20	koi_insol_err1	0.105152
8	koi_model_snr	0.142149	21	koi_srad	0.102525
9	koi_srad_err1	0.132922	22	koi_insol	0.102372
10	koi_time0bk_err1	0.131132	23	koi_teq	0.100868
11	koi_period	0.128600	24	koi_impact	0.099700
12	koi_slogg_err2	0.123361	25	dec	0.099228
13	koi_time0bk_err2	0.123268			

Therefore, our dataset for the rest of the project consists of the above variables together with our target column `koi_disposition`.

3.7 Sampling

Finally, because we still have 7316 examples, we decided to do reduce the size of the dataset to 2000 entries through sampling. We did it mainly due to performance issues, to make the dataset more manageable and reduce execution times. We also checked that the resulting sampled dataset respected the proportions of our target column, and we obtained a very similar one.

4 Evaluation criteria of data mining models

To obtain a representative validation data set, we decided to split the dataset 70% for training and 30% for validation and testing. We also checked that the target column categories had their proportions respected. However, we didn't create training and validation datasets CVS's as such. Instead in each algorithm we performed `train_test_split` with the same seed and parameters.

To optimize the parameters for the algorithms that required it we used k-fold cross validation. The value of k chosen for k-fold cross validation is 5. We selected it mainly because of computational performance and execution time issues.

Regarding the metrics, we decided to use f1-score to asses the quality of our classifiers. We choose it mainly because our dataset is slightly unbalanced, with a 30:70 split in our target column, and therefore f1-score provided clearer understanding of the performance and the behaviour of the algorithms.

5 Machine learning methods

5.1 Naïve Bayes

Naïve Bayes works on the assumption of independence between variables in the dataset. First of all we checked weather or not this is a reasonable assumption in our dataset. To do so we calculated the correlation matrix, the farther apart the correlations are from 0 the more dependence there is between variables.

As we see in appendix B despite seeing some strong correlations between pairs of variables, for most pairs these values are much closer to 0 than expected. Therefore it isn't that far-fetched to assume independence between variables.

There are several types of Naïve Bayes algorithms:

- Gaussian NB: Used when feature space is quantitative
- Bernoulli NB: Used when feature space is Binary
- Multinomial NB: Used when feature space is discrete counts

Our data consists mostly of numerical quantitative variables, therefore Gaussian Naïve Bayes will be applied from here on.

5.2 Normalization

We executed the algorithm with no preprocessing, with standardization (mean=0, std=1) and we also tried Power Transform, which is an algorithm that tries to make our data more Gaussian-like.

	No normalization	Standardization	Power Transform
Confusion matrix	$\begin{bmatrix} 208 & 200 \\ 3 & 189 \end{bmatrix}$	$\begin{bmatrix} 287 & 121 \\ 2 & 190 \end{bmatrix}$	$\begin{bmatrix} 357 & 51 \\ 19 & 173 \end{bmatrix}$
Accuracy:	0.661	0.795	0.883
F1 score:	0.65	0.755	0.831

One of the problems we had when performing Naïve Bayes is that most of our data are continuous variables which don't exactly follow a normal distribution. Gaussian Naïve Bayes will treat out data as if they followed a Gaussian distribution. That is why the f1-score and accuracy of the Power transformed data is much better than the others.

5.3 Parameter tuning

Naïve Bayes doesn't have many hyperparameters, in fact we will only analyse `var_smoothing` which determines the amount Laplace smoothing applied.

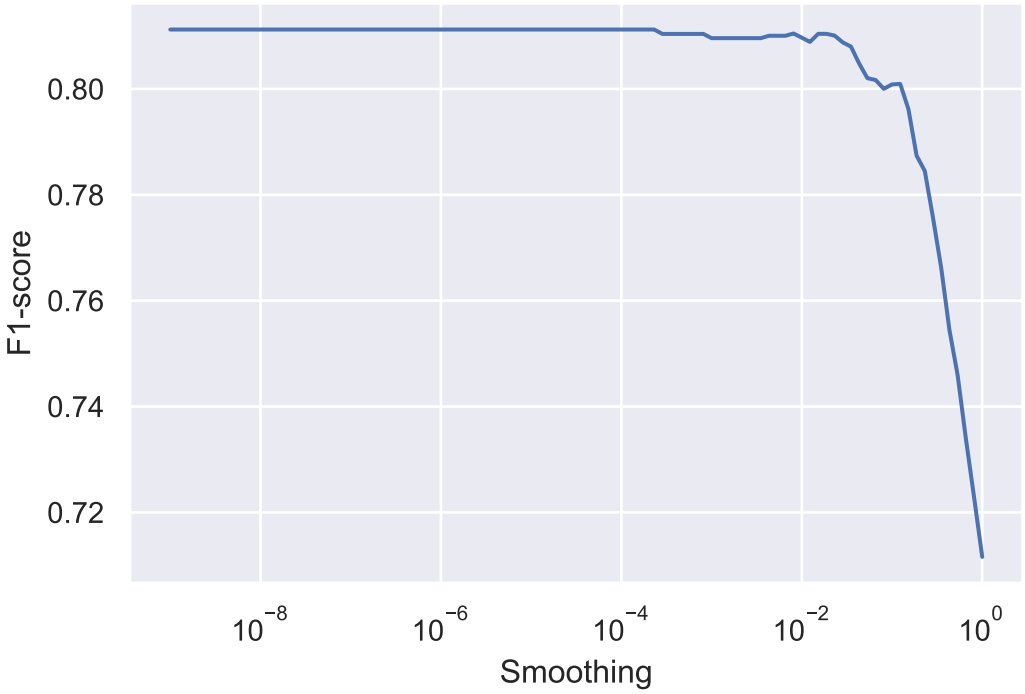


Figure 2: Naïve Bayes smoothing

Looking at the image we can see that the f1-score barely changes when the `var_smoothing` is in the range $[10^{-9}, 10^{-2}]$. Therefore its better to leave the default 10^{-9}.

5.4 K-NN

To make our KNN performance a little better we tried different dimensionality reduction techniques and compared their results. The three candidates were PCA, that's the most common and the one we've seen in class, LDA, that's closely related to PCA but actively tries to find differences between the classes and finally NCA, which is a more complex method that could actually substitute KNN and uses a pretty related algorithm called stochastic nearest neighbours.

We tried values of K ranging from 1 to 50 with both weighted and unweighted KNN for the three dimensionality reduction methods and got this output:

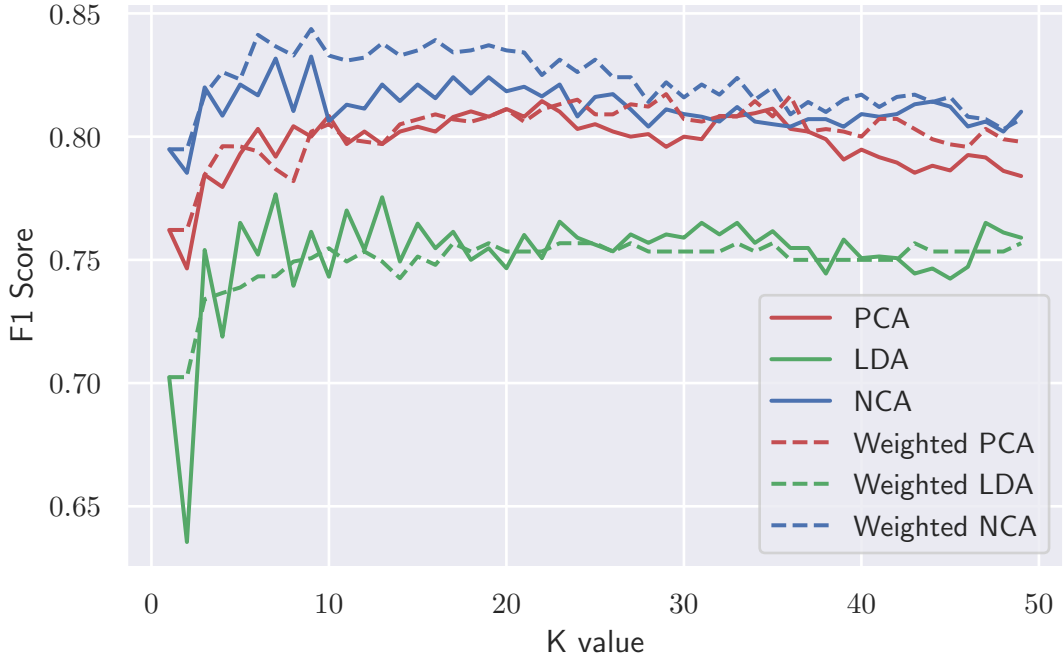


Figure 3: weighted and unweighted knn with PCA, LDA and NCA

Confusion matrix on test set:	$\begin{bmatrix} 367 & 41 \\ 22 & 170 \end{bmatrix}$
Accuracy on test set:	0.895
F1 score on test set:	0.844

As we see, NCA gets the best result, and the weighted variant of it goes a little bit further giving us even better results, as for the K value, we see that the f1 score is bad for very low values of K, gets maximized at K≈9 and then it stays stable for the tested range.

5.5 Decision Trees

We decided that in order to find the ideal depth of the decision tree, we had to try a range of values and pick the best performing one.

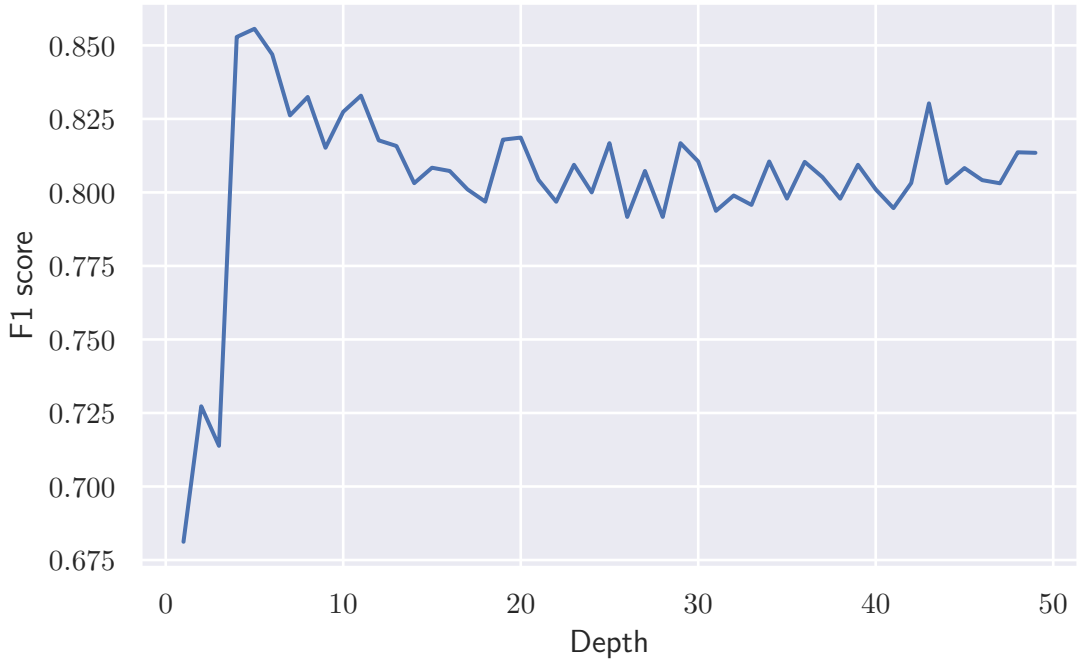


Figure 4: Decision trees F1 Score depending on the maximum depth of the decision tree.

As we see, we tried all values between 1 and 50 and we found out that the best performing depth was around 5, that had the following confusion matrix:

$$\text{Confusion matrix on test set: } \begin{bmatrix} 382 & 26 \\ 29 & 163 \end{bmatrix}$$

Accuracy on test set: 0.908
F1 score on test set: 0.856

The resulting tree was surprisingly simple, and we could find very good leafs, with a gini index of 0 and 300+ samples in it and also a pretty bad leaf that had a gini value of 0.5 but just 34 samples. The majority of the examples though were classified by two leafs that had a really small gini and a very big number of samples, that'd make it a pretty good classifier.

To explain in a simple way what the tree does, we could say that it first tries to find the examples that are clearly a False Positive with the rightmost branches and after that, tries to classify the remaining ones. We can understand that this behaviour is reinforced by the nature of our data, that's skewed towards the False Positive class.

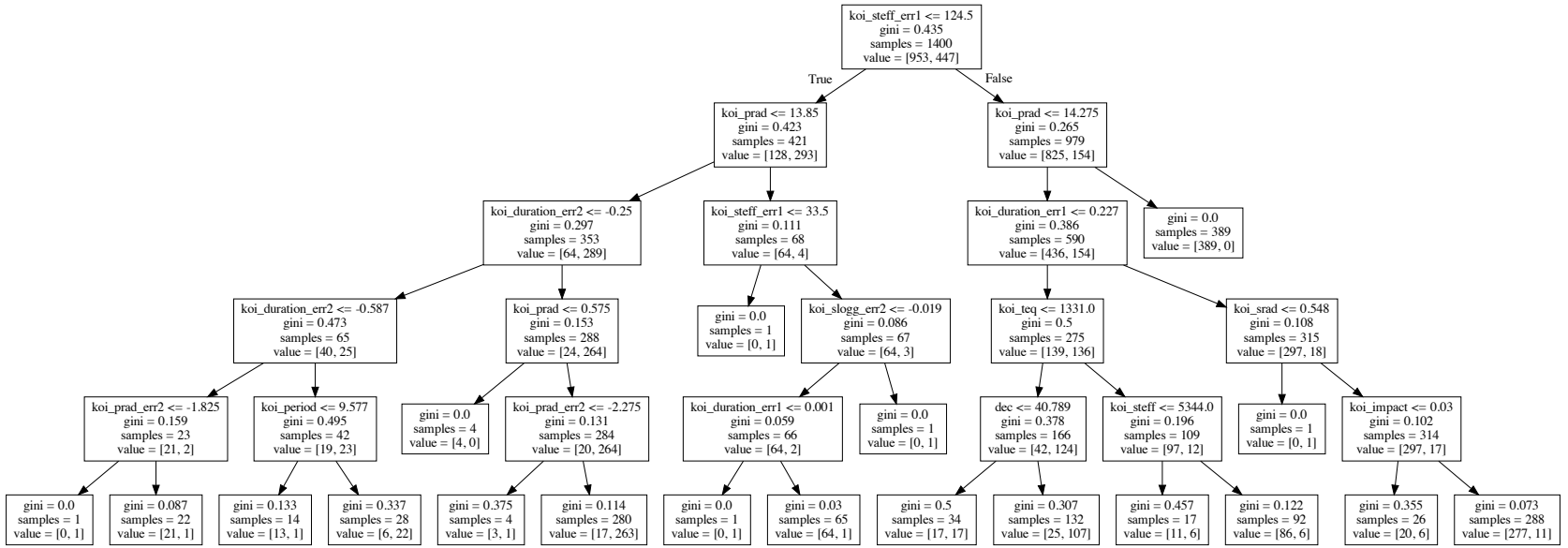


Figure 5: Our best-performing decision tree.

5.6 Support vector Machines

5.7 Comparison between kernels

We tried various kernels and performed a grid search for the best parameters for each one. When doing the search, we performed a 5-fold cross validation since with 10 folds it took too much time and the results where fairly similar. We also tried to reduce the number of features to 12 (from the 25 we selected in section 3.6) to reduce execution time but the results where significantly worse (around 0.15 less accuracy).

As shown in table 3, the best results where obtained using the linear kernel, although there is not much difference between them. The main difference is the percentage of supports which is notably smaller in the linear kernel. In the following sections we show in detail the best results obtained with every kernel and a plot of the parameter search.

Table 3: Comparison of different SVM kernels

Kernel	Accuracy	F1	Supports	Proportion	C	γ
Linear	0.9150	0.8668	288(269)	0.2057	10^5	
Polynomial 2	0.9067	0.8542	332(283)	0.2371	10^4	
Polynomial 3	0.9050	0.8503	356(317)	0.2543	10^3	
RBF	0.9083	0.8564	330(302)	0.2357	10^6	0.001

5.8 Linear SVM

With default parameters we obtained the following results:

Confusion matrix on test set:	$\begin{bmatrix} 387 & 21 \\ 54 & 138 \end{bmatrix}$
Accuracy on test set:	0.875
F1 score on test set:	0.7863

By tuning the C parameter as show in fig. 6 we managed to get much better results.

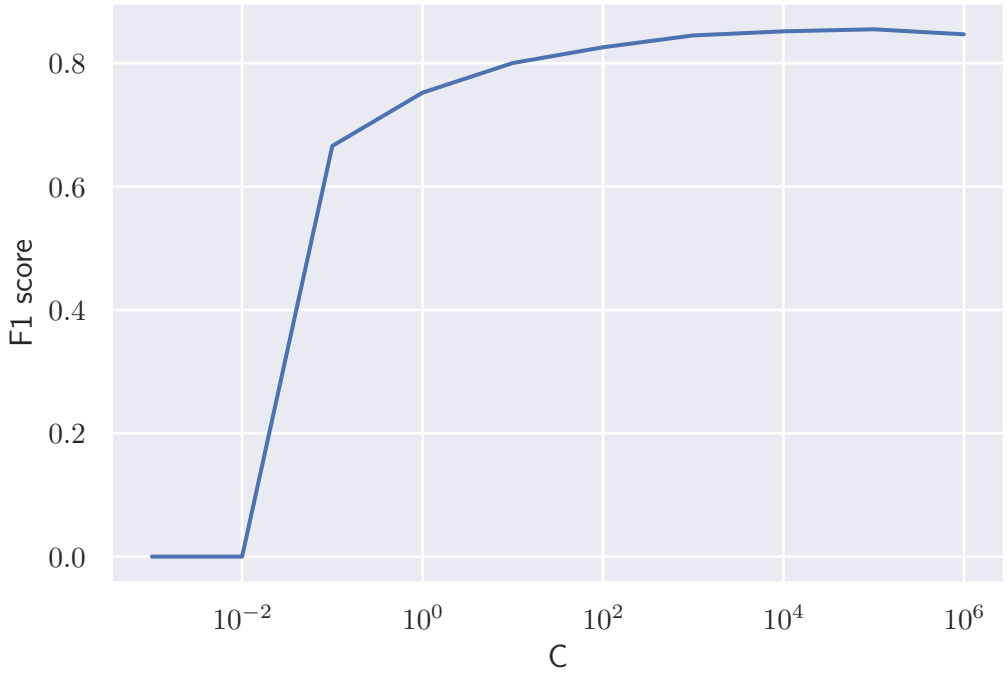


Figure 6: linear SVM C parameter search

Best results ($C = 10^5$)	
Confusion matrix on test set:	$\begin{bmatrix} 383 & 25 \\ 26 & 166 \end{bmatrix}$
Accuracy on test set:	0.915
F1 on test set:	0.8668
Number of supports:	288 (269 of them have slacks)
Proportion of supports:	0.2057

We can see that the number of false positives in our confusion matrix halved with respect to the default value. We achieved an accuracy of 0.915 and the proportion of supports is only 20%

5.9 Polynomial SVM

Best results ($C = 10^4$)	
Confusion matrix on test set:	$\begin{bmatrix} 380 & 28 \\ 28 & 164 \end{bmatrix}$
Accuracy on test set:	0.9067
F1 on test set:	0.8542
Number of supports:	332 (283 of them have slacks)
Proportion of supports:	0.2371

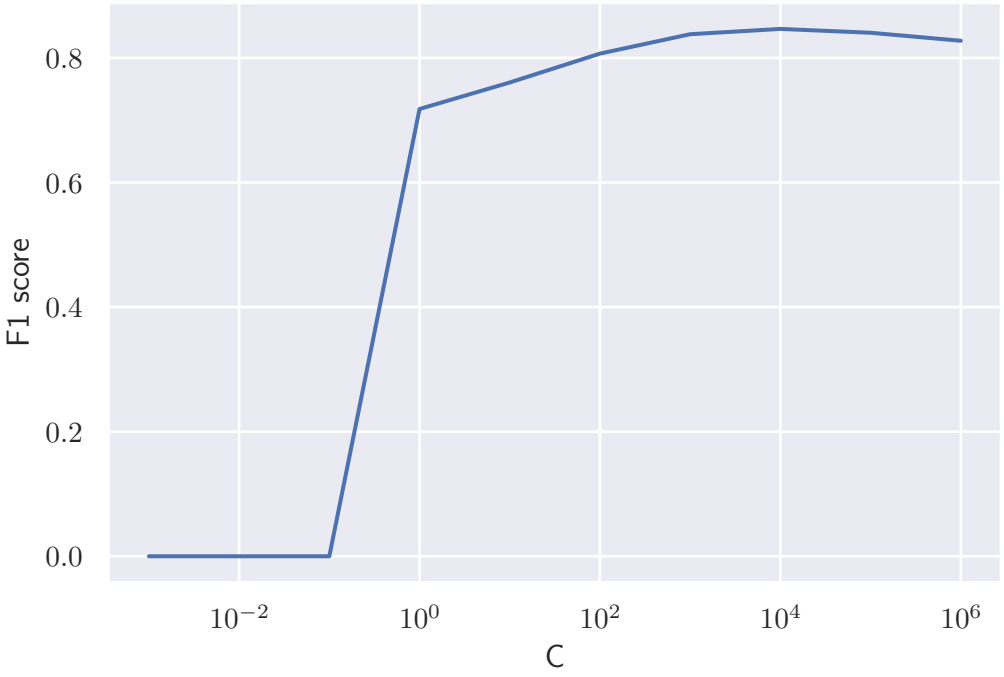


Figure 7: 2nd degree polynomial SVM C parameter search

Best results ($C = 10^3$)	
Confusion matrix on test set:	$\begin{bmatrix} 381 & 27 \\ 30 & 162 \end{bmatrix}$
Accuracy on test set:	0.905
F1 on test set:	0.8503
Number of supports:	356 (317 of them have slacks)
Proportion of supports:	0.2543

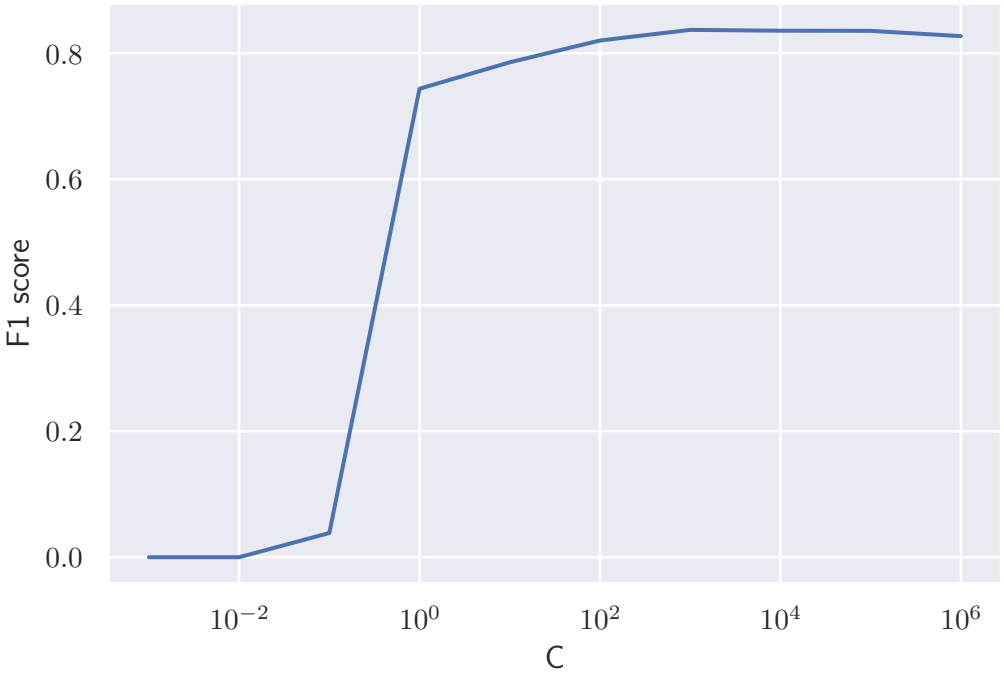


Figure 8: 3rd degree polynomial SVM C parameter search

We obtained better results with the second degree polynomial although they both have similar accuracy. The linear kernel outperformed both of them and had less support vectors.

5.10 RBF SVM

With RBF we obtain better accuracy than with polynomial kernels but still not as good as with the linear kernel.

Best results ($C = 10^6, gamma = 0.001$)	
Confusion matrix on test set:	$\begin{bmatrix} 381 & 27 \\ 28 & 164 \end{bmatrix}$
Accuracy on test set:	0.9083
F1 on test set:	0.8564
Number of supports:	330 (302 of them have slacks)
Proportion of supports:	0.2357

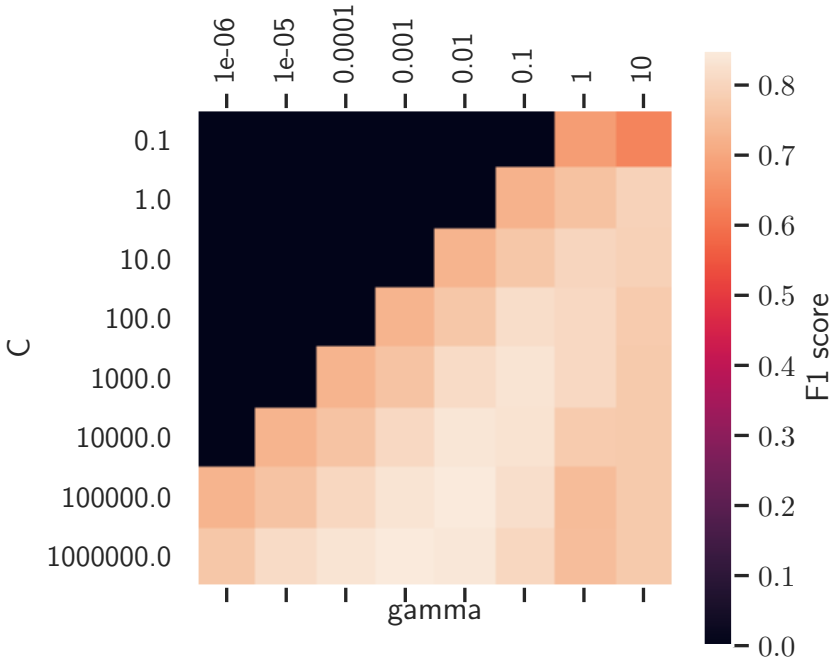


Figure 9: RBF SVM C parameter search

5.11 Meta-learning algorithms

5.11.1 Performance Majority Voting

Majority voting uses several of the algorithms already commented in this project. Therefore we will use those algorithms with the best combination of preprocessing and hyperparameters we already found.

Table 4: Majority voting results

Method	Accuracy
Naïve Bayes	0.884
K-NN	0.857
Decision Tree	0.877
Majority voting	0.914
Majority voting (weighted)	0.914

With hard voting:

Confusion matrix on test set: $\begin{bmatrix} 375 & 33 \\ 20 & 172 \end{bmatrix}$

Accuracy on test set: 0.9117
F1 score on test set: 0.8622

With weighted voting (2 1 2):

Confusion matrix on test set: $\begin{bmatrix} 373 & 35 \\ 19 & 173 \end{bmatrix}$

Accuracy on test set: 0.9100
F1 score on test set: 0.8492

There is no benefit to weighting the votes since all 3 methods offer very similar accuracy.

5.11.2 Bagging

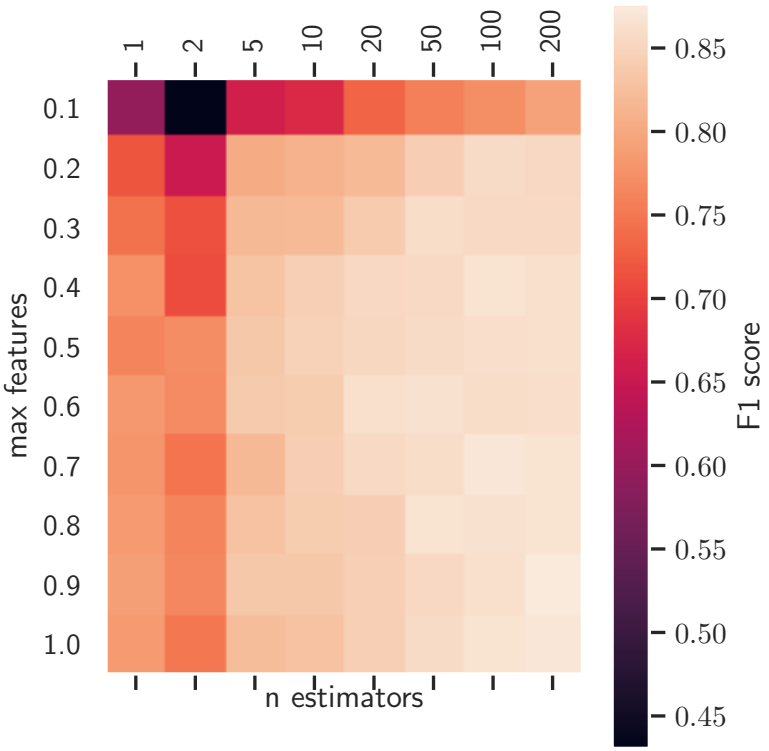


Figure 10: Bagging parameter search

The best results where obtained with: `n_est` $= 200$, `max_features` $= 0.9$. Nonetheless, as we can see in fig. 10 results with `n_est` ≥ 50, `max_features` ≥ 0.3 are pretty similar.

Confusion matrix on test set:	$\begin{bmatrix} 388 & 20 \\ 24 & 168 \end{bmatrix}$
Accuracy on test set:	0.9267
F1 score on test set:	0.8825

5.11.3 RandomForest

Random Forest is an algorithm based in the combination of multiple decision trees. It has many hyperparameters among which we decided to optimize the 6 following: `bootstrap`, `max_features`, `max_features`, `min_samples_leaf`, `min_samples_split`, `n_estimators`. At the beginning we wanted to have a pool of many values per parameter. However the time of computation was too high and we ended up doing a 5-fold with 2-3 values per parameter.

Table 5: RandomForest best parameters

Parameter	Value
`bootstrap`	True
`max_depth`	150
`max_features`	10
`min_samples_leaf`	5
`min_samples_split`	5
`n_estimators`	500

Confusion matrix on test set: $\begin{bmatrix} 384 & 24 \\ 21 & 171 \end{bmatrix}$

Accuracy on test set: 0.925
F1 score on test set: 0.883

5.11.4 Adaboost

The final algorithm we implemented is Ada boosting. The base implementation of this algorithms works with decision trees as it's classifier. This type of boosting basically consists on performing different executions of the classifier but changing the weights. The algorithm has two additional hyperparameters `n_estimators` and `learning_rate`.

After that we fined tuned the parameters and we found that the best combination is `n_estimators` = 60 and `learning_rate` = 0.5

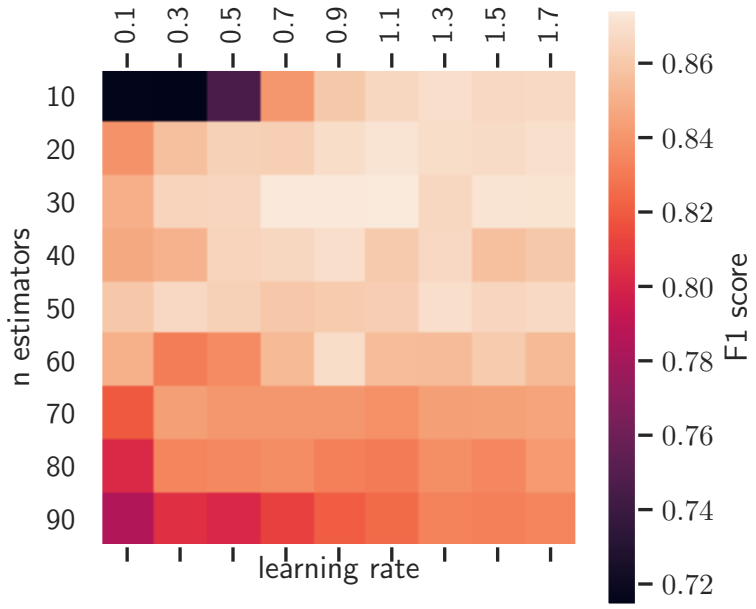


Figure 11: AdaBoost parameter search

When executing Adaboost with the best parameters found we obtain the following results:

Confusion matrix on test set:	$\begin{bmatrix} 383 & 25 \\ 21 & 171 \end{bmatrix}$
Accuracy on test set:	0.923
F1 score on test set:	0.881

6 Comparison

Once we have all the algorithms implemented we are going to compare them to find out which yields the best results in our dataset. To do so we run the algorithms with their respective best parameters on the testing dataset and we get the following table:

Table 6: Comparison of metrics

	Accuracy	F1
nb	0.883333	0.831731
knn	0.878333	0.814249
dt	0.910000	0.858639
svm	0.911667	0.865140
voting	0.910000	0.860825
bag	0.920000	0.874346
rf	0.923333	0.878947
ada	0.923333	0.881443

Looking at table 6 we can see that meta learning algorithms have the best f1-scores and accuracies. Ada boosting is tied with random forests for the best accuracy but it has a slightly higher f1-score. Another thing to point out is that knn is the worse performing algorithm. Despite all of this all the algorithms have relatively similar results, we only have a 0.4% accuracy and a 0.5% f1-score difference between the best and worst.

Next we continue the comparison by performing chi-squared mcNemar tests among all algorithms to find out whether or not there is a meaningful difference between them.

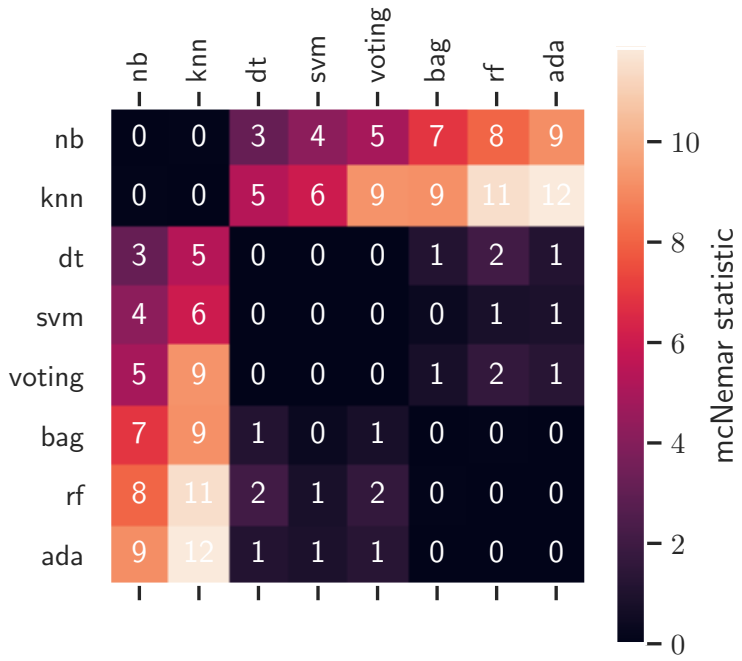


Figure 12: mcNemar statistic

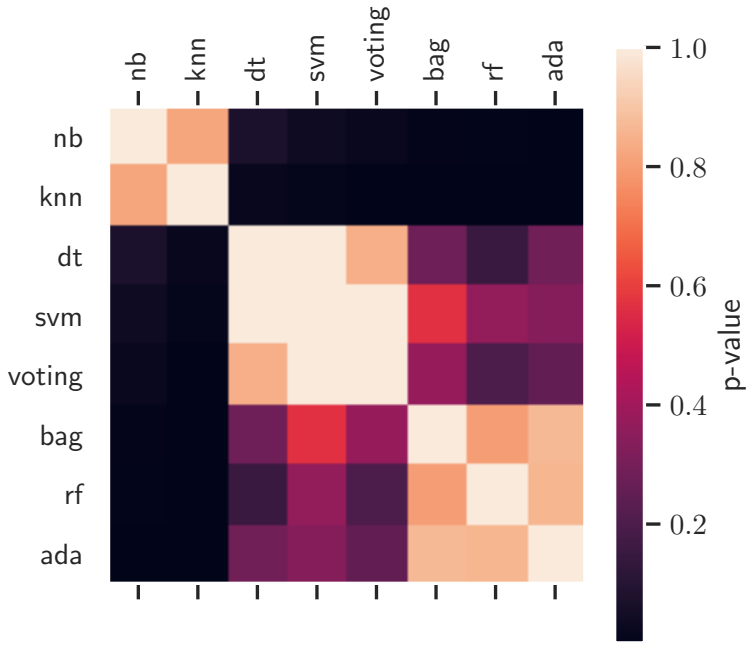


Figure 13: mcNemar test p-value

Table 7: mcNemar test p-values

	nb	knn	dt	svm	voting	bag	rf	ada
nb	1.00	0.82	0.08	0.04	0.03	0.01	0.00	0.00
knn	0.82	1.00	0.02	0.01	0.00	0.00	0.00	0.00
dt	0.08	0.02	1.00	1.00	1.00	0.29	0.15	0.29
svm	0.04	0.01	1.00	1.00	1.00	0.57	0.37	0.34
voting	0.03	0.00	1.00	1.00	1.00	0.38	0.20	0.26
bag	0.01	0.00	0.29	0.57	0.38	1.00	0.80	0.87
rf	0.00	0.00	0.15	0.37	0.20	0.80	1.00	1.00
ada	0.00	0.00	0.29	0.34	0.26	0.87	1.00	1.00

When analyzing the p-values obtained by the mcNemar test we want to have 95% of confidence level, therefore we will set $\alpha = 0.05$. If we get $p > \alpha$ we fail to reject the $H0$. Otherwise if $p \leq \alpha$ we reject $H0$. When looking at table 7 we can clearly see that Naïve Bayes and Knn have p-values smaller than alpha when tested against any other algorithm. Therefore we can reject H0 and conclude that they have significantly different performance than the rest. When analysing other algorithms we fail to reject the null hypothesis and therefore their performance is relatively similar.

Naïve Bayes and Knn had the worst scoring results table 6. Therefore we end up concluding that choosing any of the other algorithms will give better results. As they aren't significantly different between them the choice to use one or the other depends individual preference and computational cost.

References

[1] Administrator, NASA. *New NASA Kepler Mission Data*. NASA. Publisher: Brian Dunbar. June 7, 2013. URL: http://www.nasa.gov/mission_pages/kepler/news/kepler-new-data-q-and-a.html (visited on 12/20/2020).

[2] Archive, NASA Exoplanet. *Kepler Objects of Interest Cumulative Table*. type: dataset. 2019. DOI: 10.26133/NEA4. URL: https://catcopy.ipac.caltech.edu/dois/doi.php?id=10.26133/NEA4 (visited on 12/20/2020).

[3] *Exoplanet Archive Documentation*. URL: https://exoplanetarchive.ipac.caltech.edu/applications/DocSet/index.html?doctree=/docs/docmenu.xml&startdoc=1 (visited on 12/20/2020).

[4] *Kepler Exoplanet Search Results*. URL: https://kaggle.com/nasa/kepler-exoplanet-search-results (visited on 12/20/2020).

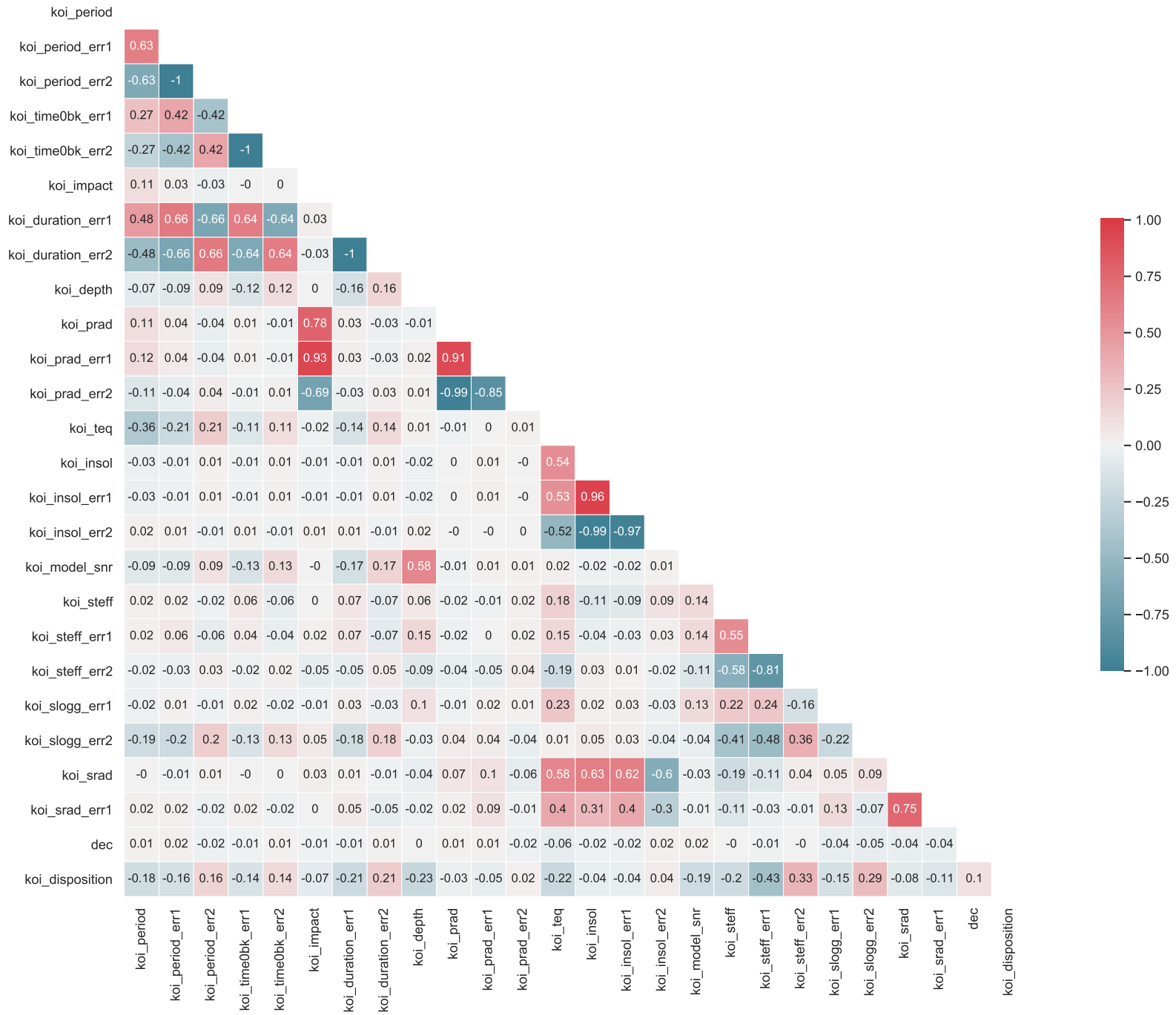
[5] Pedregosa, F. et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] *statsmodels.stats.contingency_tables.mcnemar — statsmodels*. URL: https://www.statsmodels.org/dev/generated/statsmodels.stats.contingency_tables.mcnemar.html (visited on 12/23/2020).

A Feature score table

Table 8: Score of all features

Variable	Score
koi_steff_err1	0.193758
koi_prad	0.184690
koi_steff_err2	0.170011
koi_prad_err1	0.164256
koi_duration_err2	0.156999
koi_prad_err2	0.154296
koi_duration_err1	0.153266
koi_model_snr	0.142149
koi_srad_err1	0.132922
koi_time0bk_err1	0.131132
koi_period	0.128600
koi_slogg_err2	0.123361
koi_time0bk_err2	0.123268
koi_slogg_err1	0.120881
koi_steff	0.115671
koi_period_err2	0.109981
koi_period_err1	0.108709
koi_insol_err2	0.108357
koi_depth	0.106608
koi_insol_err1	0.105152
koi_srad	0.102525
koi_insol	0.102372
koi_teq	0.100868
koi_impact	0.099700
dec	0.099228
ra	0.096678
koi_srad_err2	0.092219
koi_depth_err2	0.091970
koi_depth_err1	0.090008
koi_slogg	0.088974
koi_kepmag	0.077437
koi_time0bk	0.072244
koi_impact_err2	0.061436
koi_impact_err1	0.061006
koi_tce_plnt_num	0.052011
koi_duration	0.022304


koi_period
koi_period_err1 0.63
koi_period_err2 -0.63 -1
koi_time0bk_err1 0.27 0.42 -0.42
koi_time0bk_err2 -0.27 -0.42 0.42 -1
koi_impact 0.11 0.03 -0.03 -0 0
koi_duration_err1 0.48 0.66 -0.66 0.64 -0.64 0.03
koi_duration_err2 -0.48 -0.66 0.66 -0.64 0.64 -0.03 -1
koi_depth -0.07 -0.09 0.09 -0.12 0.12 0 -0.16 0.16
koi_prad 0.11 0.04 -0.04 0.01 -0.01 0.78 0.03 -0.03 -0.01
koi_prad_err1 0.12 0.04 -0.04 0.01 -0.01 0.93 0.03 -0.03 0.02 0.91
koi_prad_err2 -0.11 -0.04 0.04 -0.01 0.01 -0.69 -0.03 0.03 0.01 -0.99 -0.85
koi_teq -0.36 -0.21 0.21 -0.11 0.11 -0.02 -0.14 0.14 0.01 -0.01 0 0.01
koi_insol -0.03 -0.01 0.01 -0.01 0.01 -0.01 -0.01 0.01 -0.02 0 0.01 -0 0.54
koi_insol_err1 -0.03 -0.01 0.01 -0.01 0.01 -0.01 -0.01 0.01 -0.02 0 0.01 -0 0.53 0.96
koi_insol_err2 0.02 0.01 -0.01 0.01 -0.01 0.01 0.01 -0.01 0.02 -0 -0 0 -0.52 -0.99 -0.97
koi_model_snr -0.09 -0.09 0.09 -0.13 0.13 -0 -0.17 0.17 0.58 -0.01 0.01 0.01 0.02 -0.02 -0.02 0.01
koi_steff 0.02 0.02 -0.02 0.06 -0.06 0 0.07 -0.07 0.06 -0.02 -0.01 0.02 0.18 -0.11 -0.09 0.09 0.14
koi_steff_err1 0.02 0.06 -0.06 0.04 -0.04 0.02 0.07 -0.07 0.15 -0.02 0 0.02 0.15 -0.04 -0.03 0.03 0.14 0.55
koi_steff_err2 -0.02 -0.03 0.03 -0.02 0.02 -0.05 -0.05 0.05 -0.09 -0.04 -0.05 0.04 -0.19 0.03 0.01 -0.02 -0.11 -0.58 -0.81
koi_slogg_err1 -0.02 0.01 -0.01 0.02 -0.02 -0.01 0.03 -0.03 0.1 -0.01 0.02 0.01 0.23 0.02 0.03 -0.03 0.13 0.22 0.24 -0.16
koi_slogg_err2 -0.19 -0.2 0.2 -0.13 0.13 0.05 -0.18 0.18 -0.03 0.04 0.04 -0.04 0.01 0.05 0.03 -0.04 -0.04 -0.41 -0.48 0.36 -0.22
koi_srad -0 -0.01 0.01 -0 0 0.03 0.01 -0.01 -0.04 0.07 0.1 -0.06 0.58 0.63 0.62 -0.6 -0.03 -0.19 -0.11 0.04 0.05 0.09
koi_srad_err1 0.02 0.02 -0.02 0.02 -0.02 0 0.05 -0.05 -0.02 0.02 0.09 -0.01 0.4 0.31 0.4 -0.3 -0.01 -0.11 -0.03 -0.01 0.13 -0.07 0.75
dec 0.01 0.02 -0.02 -0.01 0.01 -0.01 -0.01 0.01 0 0.01 0.01 -0.02 -0.06 -0.02 -0.02 0.02 0.02 -0 -0.01 -0 -0.04 -0.05 -0.04 -0.04
koi_disposition -0.18 -0.16 0.16 -0.14 0.14 -0.07 -0.21 0.21 -0.23 -0.03 -0.05 0.02 -0.22 -0.04 -0.04 0.04 -0.19 -0.2 -0.43 0.33 -0.15 0.29 -0.08 -0.11 0.1
koi_period
koi_period_err1
koi_period_err2
koi_time0bk_err1
koi_time0bk_err2
koi_impact
koi_duration_err1
koi_duration_err2
koi_depth
koi_prad
koi_prad_err1
koi_prad_err2
koi_teq
koi_insol
koi_insol_err1
koi_insol_err2
koi_model_snr
koi_steff
koi_steff_err1
koi_steff_err2
koi_slogg_err1
koi_slogg_err2
koi_srad
koi_srad_err1
dec
koi_disposition
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00